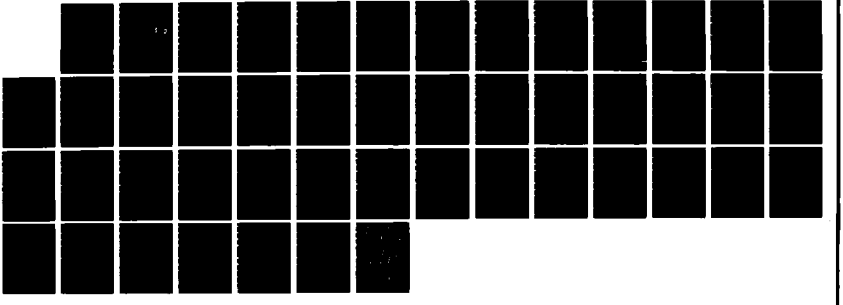
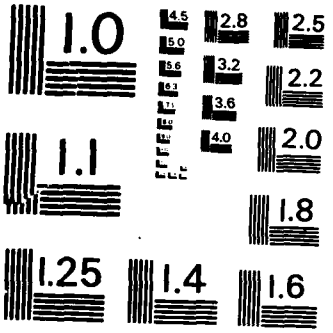
AD-A166 658 MACROMODELING AND OPTIMIZATION OF DIGITAL MOS VLSI CIRCUITS(U) MASSACHUSETTS INST OF TECH CAMBRIDGE RESEARCH LAB OF ELECTRONICS M D MATSON ET AL. MAR 86 1/1

UNCLASSIFIED VLSI-MEMO-86-303 N00014-80-C-0622 F/G 9/5 NL





VLSI Memo No. 86-303
March 1986

MACROMODELING AND OPTIMIZATION OF DIGITAL MOS VLSI CIRCUITS

Mark D. Matson and Lance A. Glasser

Abstract

Power consumption and signal delay are crucial to the design of high-performance VLSI circuits. This paper presents CAD tools for modeling and optimizing digital MOS designs. The tools determine the transistor sizes that minimize circuit power consumption subject to constraints on signal path delays. Computational efficiency is obtained through macromodeling techniques and a specialized optimization algorithm. The macromodels are based on device equations, and encapsulate logic gate behavior in a set of simple yet accurate formulas. The optimization algorithm exploits properties of the digital MOS domain to convert the primal optimization problem into a dual form which is much easier to solve. The result is a pair of CAD tools that can optimize a circuit in roughly the amount of time needed to perform a transistor level simulation of the circuit.

Microsystems Research Center
Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

Acknowledgements

This research was supported in part by an RCA fellowship, by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622, and by the Air Force under contract number F49620-84-C-0004.

Author Information

Matson, current address: Symbolics, 11 Cambridge Center, Cambridge, MA 02142; Glasser: Research Laboratory of Electronics and the Department of Electrical Engineering and Computer Science, MIT, Room 36-880, Cambridge, MA 02139, (617) 253-4677.

Table of Contents

1. Introduction 1
2. Overview of the Paper 3
3. Macromodeling 3
    3.1. Introduction 3
    3.2. Motivation and Intent 4
    3.3. Inverters 6
        3.3.1. Objective Function 6
        3.3.2. Output Waveform 7
        3.3.3. Input Capacitance 14
    3.4. General Logic Gates 18
        3.4.1. Output Waveform 18
        3.4.2. Input Capacitance 22
    3.5. Implementation 24
4. Optimization 26
    4.1. Introduction 26
    4.2. Properties of Our Problem 26
    4.3. Duality 28
        4.3.1. Lagrange Multipliers 29
        4.3.2. Finding the Optimum 30
        4.3.3. Degenerate Cases 34
        4.3.4. Restrictions 35
    4.4. Implementation 35
5. Future Work 37
6. Acknowledgments 39
References 40

List of Figures

Figure 1: Waveform Characterization 5
Figure 2: Macromodel Representation 6
Figure 3: A Depletion Load nMOS Inverter 6
Figure 4: Delay Mapping 9
Figure 5: Circuit for Determining Drive Curves 9
Figure 6: Comparison of Drive Curves 10
Figure 7: Inverter's Predicted Response 12
Figure 8: Inverter's Actual Output Responses 13
Figure 9: Input Capacitance Model 14
Figure 10: Expected Input Capacitance 15
Figure 11: Capacitance for a Falling Input 16
Figure 12: Capacitance for a Rising Input 17
Figure 13: General Logic Gate 18
Figure 14: Example of a General Logic Gate 19
Figure 15: Circuit Model for a General Logic Gate 20
Figure 16: Reduction of Effective Transconductance 20
Figure 17: Circuit Model for Input Capacitance 23
Figure 18: Input Capacitance for the Top Input of a NAND Gate 25
Figure 19: Input Capacitance for the Bottom Input of a NAND Gate 25
Figure 20: Contours of Delay and Power 30
Figure 21: Set of Possible Pairs 31
Figure 22: Reaching the Optimum 31
Figure 23: Inner Loop Minimization 32
Figure 24: Outer Loop Maximization 34
Figure 25: A Full Adder Module 38

List of Tables

Table 1: Pullup/Pulldown Regions of Operation 8
Table 2: Macromodel Curve Fit Accuracies 26
Table 3: Optimization Statistics for the Inverter Chain 36
Table 4: Optimization Statistics for the Four Bit Adder 39

Macromodeling and Optimization of Digital MOS VLSI Circuits[1]

Mark D. Matson
Symbolics
11 Cambridge Center
Cambridge, Massachusetts 02142

Lance A. Glasser
Research Laboratory of Electronics
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

Power consumption and signal delay are crucial to the design of high-performance VLSI circuits. This paper presents CAD tools for modeling and optimizing digital MOS designs. The tools determine the transistor sizes that minimize circuit power consumption subject to constraints on signal path delays. Computational efficiency is obtained through macromodeling techniques and a specialized optimization algorithm. The macromodels are based on device equations, and encapsulate logic gate behavior in a set of simple yet accurate formulas. The optimization algorithm exploits properties of the digital MOS domain to convert the primal optimization problem into a dual form which is much easier to solve. The result is a pair of CAD tools that can optimize a circuit in roughly the amount of time needed to perform a transistor level simulation of the circuit.

1. Introduction

The design of a VLSI circuit is an enormous task. Sophisticated CAD tools are essential if designers are to take full advantage of the power offered by fabrication technology. We describe a tool for optimizing the performance of digital MOS circuits. This tool finds the transistor sizes that minimize power consumption subject to constraints on signal path delays. The principal advantage is an increase in designer productivity. At present, designers size transistors based on intuition and numerous SPICE simulations. This process is so time consuming—for both man and machine—that designers are hard-pressed to arrive at any circuit that meets delay specifications and can rarely afford the extra effort needed to minimize power consumption as well. This hinders not only the design of the circuit at hand, but also the comparison of alternate topologies

[1]This research was supported by an RCA fellowship, the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622, and the Air Force under contract number F49620-84-C-0004.

for implementing functional blocks, as the performance benefits offered by different topologies cannot be truly ascertained unless the corresponding circuits have been optimized.

Another application is automatic module generation for silicon compilers. The module's transistors must be properly sized in order to meet system performance specifications, but it would be unthinkable to have a human perform the sizing. The task could involve thousands of transistors, making it too mundane and complicated. A special purpose optimizer can accomplish the chore far more efficiently.

Several authors have studied optimization work of this nature. General purpose optimization packages such as DELIGHT [1] and APLSTAP [2] perform much of the work in the optimization process. They iteratively improve the design solution as a designer would, but by employing nonlinear optimization algorithms, choose the next solution point more accurately and efficiently than a human could. The key advantage is that an optimal solution is reached. However the optimization process tends to be computationally expensive for a number of reasons. First, since the optimization package is general purpose in nature, it cannot exploit properties of digital MOS logic and use algorithms which would be more problem specific and hence potentially faster. Second, because the optimization package is isolated from the circuit's data base, communicating solely via the simulator, there is no mechanism to access the circuit's structural description or to embed additional information in the data base which could assist the optimization. This hampers the application of more efficient algorithms that would require such provisions. Third, the circuit's signal path delays must be determined fairly accurately; this generally entails the use of a device level simulator such as SPICE, which is rather expensive computationally. The consequence of these three factors is that general purpose optimizers are typically restricted to circuits with at most about thirty design parameters.

In an effort to address larger designs, some researchers have investigated more specialized techniques [3]. By using a resistive model for transistors and neglecting the changes in a logic gate's input capacitance induced by sizing its transistors, these workers were able to simplify the optimization problem greatly. They reformulated the original problem, a minimization subject to nonlinear constraints, as an unconstrained minimization. This allows for much simpler optimization algorithms, leading to fast convergence times. Nonetheless, the simplifications needed to reformulate the problem seriously reduce the accuracy of both the power minimization and the satisfaction of the delay constraints, making the approach inappropriate for high-performance circuit design.

Other authors have aimed for fast computation times by simplifying both the logic gate models and the optimization techniques. Examples are TV [4] and Andy [5]. These tools use resistor models for transistors instead of the computationally expensive device level models. Heuristics, rather than nonlinear optimization algorithms, guide the sizing of transistors in critical paths. In particular, TV speeds up paths by widening the transistors of slow logic gates, while Andy uses a fixed sizing ratio from gate to gate when a chain drives a large capacitive load. Although these approaches are computationally fast enough to be applied to large circuits, our problem domain requires more accuracy and efficiency. The resistor model is not accurate enough for high-performance design, and iteratively applying heuristics is not as efficient as nonlinear optimization algorithms that simultaneously consider all critical paths.

2. Overview of the Paper

This paper presents a novel approach to the transistor sizing problem. We attack the competing needs for accuracy, computational speed, and a nearly optimal solution by combining the benefits of the previous approaches we examined. Like TV and Andy, we work at a higher level of abstraction than SPICE, transcending the details of actual transistor operation. However we acquire additional computational speed by modeling entire logic cells rather than just individual transistors. Like the general purpose optimizers, we employ nonlinear optimization techniques. This helps to assure that we reach an optimal solution in an efficient manner. However we exploit properties of digital MOS circuits and apply a specialized algorithm to the problem, yielding striking improvements in computational speed.

The first portion of the paper discusses the theory and implementation of the logic cell macromodeler. We begin with a resistor-capacitor model and examine its limitations. We then develop a more elaborate model, one accounting for waveform shape effects. The theory gives us the form of the macromodel equations. In the implementation section we describe how the equations' parameters are determined with a sophisticated macromodeling support package. The resulting macromodels are accurate and computationally fast, and are pertinent to timing simulation and verification as well as optimization.

The second portion presents the theory and implementation of the optimization algorithms. We describe a method particularly suited to our problem, taking advantage of the properties of the digital MOS domain, and of our ability to create a circuit data base customized for the transistor sizing problem. The approach we take is called duality; it allows us to partition the optimization problem, while transforming the nonlinear delay constraints into a much simpler form. These techniques lead to very fast computation times. We close with a description of the software and several examples of the optimizer's performance.

The conclusion ends the paper with a summary of the contributions of this work and the perspectives gained from it. We also discuss several areas for future investigation, especially as related to automatic circuit design.

3. Macromodeling

3.1. Introduction

This portion of the paper discusses accurate, computationally efficient models for MOS logic gates. The models are well suited for simulation and optimization of high-performance VLSI circuits. The models are based on device equations, and acquire much of their accuracy through careful consideration of waveshape effects.

The significance of waveshape effects has been investigated by other workers. Crystal [6], a timing simulator, models transistors as resistors, but uses different values for transistor resistances depending on input waveform. While this leads to good accuracies (typically within 10% of SPICE predictions), the approach does have some limitations. For example, the tables of effective transistor resistances depend on a uniform trigger voltage (the point on a logic gate's transfer curve where $v_{OUT} = v_{IN}$) and can produce substantial errors if this restriction is removed, for instance by varying beta ratios. Moreover the table interpolations can generate jagged delay functions; this can make the optimization task more difficult.

For these reasons we chose to base our models entirely on device equations. Horowitz [7] pursued a similar strategy in modeling the delay of a MOS inverter. He derived equations for the gate's response and then obtained estimates of parameters from the gate's drive curves (curves of v_{OUT} versus v_{IN} for different values of load current).

In this portion of the paper we describe more general models. We develop equations for power consumption, output waveform, and input capacitance of a general MOS logic gate. To obtain high accuracy in the model, we wrote a macromodeling support package to determine the equations' parameters. The package curve fits the model equations to SPICE simulation results and finds the parameter set which provides the highest accuracy.

Section 2 describes the basic principles of the macromodeling approach. Section 3 presents models for MOS inverters. Our analysis opens with a treatment of the resistor-capacitor model. After studying its range of validity, we construct an improved model that accounts for waveform shape effects. The analysis is extended to more general logic gates in section 4. The theory gives us the form of the macromodel equations. In section 5 we discuss a macromodeling support package that solves for the equations' parameters.

3.2. Motivation and Intent

Circuit optimization is a computationally expensive process. It is an iterative procedure, requiring multiple simulations at each step to evaluate delays and their gradients. Moreover, high-performance circuit design requires fairly accurate delay estimates, but using a device level simulator would be out of the question for all but small circuits.

Since it is too time consuming to compute circuit responses during the optimization, we instead pursue an approach where much of the work is performed prior to the optimization. We divide a large circuit into many small pieces. This partitioning is done such that the pieces have limited, well understood interactions, while the elements inside the pieces have strong, complex interactions. Thus computing the interactions among elements within a piece would be very expensive, and it behooves us to characterize the behavior of the pieces beforehand to avoid having to compute it during the optimization.

This approach is called macromodeling. In the digital MOS domain, candidates for pieces would be cells such as logic gates and storage elements. We model the attributes of the cells as functions of the cell's internal description and boundary conditions. In particular, we are concerned with a cell's power, input load, and output waveform attributes. The cell's internal description consists of its transistor sizes, layout parasitics, and process parameters. Boundary conditions are imposed on the cell by external agents. These include input waveforms from drivers and output loading from receivers and wiring capacitances. We characterize waveforms as time-shifted ramps with exponential tails. This waveshape is representative of those found in digital MOS circuits.[2] Figure 1 displays an example. The chain of inverters is driven by a falling input waveform; the figure shows the output waveform of each gate. Here T_{BE} denotes the time shift, and T_{SW} the time constant of the exponential portion. Conceptually T_{BE} is the time until the

[2] Actual circuit waveforms begin more smoothly than our approximation. However the error is negligible because the logic gate driven by the waveform does not really begin to switch until the waveform approaches the trigger point voltage (the point on the dc transfer curve where $v_{IN} = v_{OUT}$) and is therefore insensitive to the shape of the first part of the waveform.

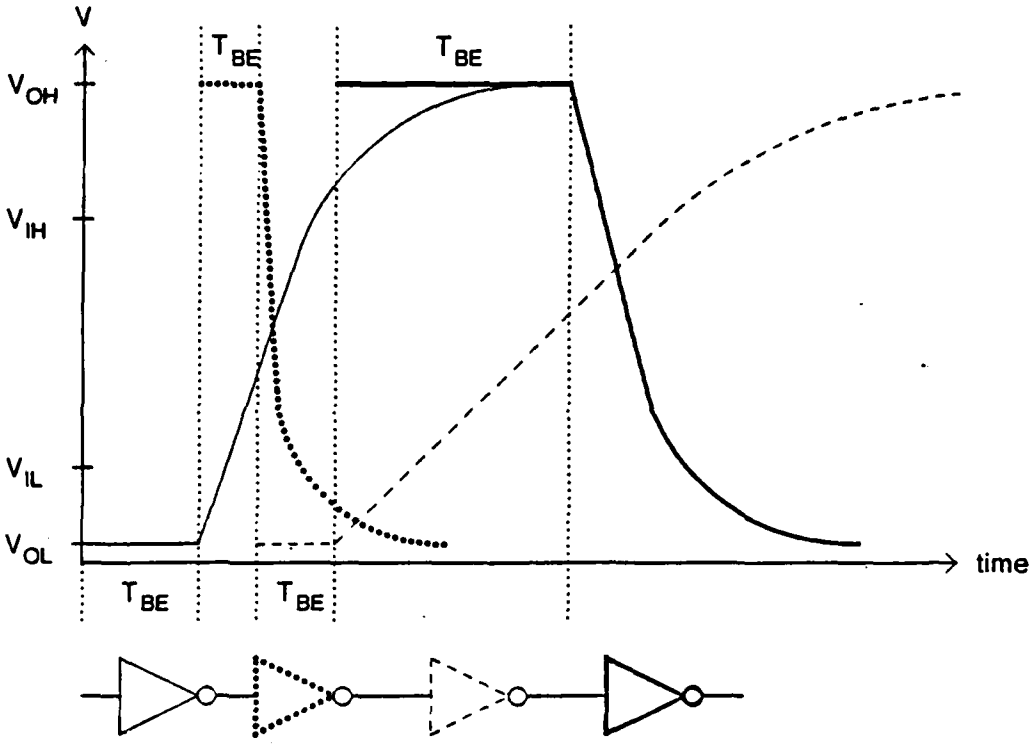


Figure 1: Waveform Characterization

output begins to move in response to an input transition, and T_{SW} is a measure of how quickly the output switches once it does begin to change. We curve fit actual circuit waveforms to the time-shifted ramps with exponential tails. From the figure we see that the output waveform of the chain of inverters is described by

$$chain\,T_{BEout} = \sum_{i=1}^{n} T_{BEout\,i}$$

$$chain\,T_{SWout} = T_{SWout\,n}$$

We characterize output loads in terms of an effective capacitance, dividing charge transferred by change in voltage. This allows us to model RC interconnection networks, since the effective capacitance can be a function of waveform slope.

We "black-box" the cell as shown in Figure 2. The cell is affected by its environment via the boundary conditions T_{SWin} and C_L. It interacts with its neighbors via its interface attributes C_{in} and T_{SWout}. The internal attributes power and T_{BEout} are isolated from the environment and have no influence on the attributes of the cell's neighbors.

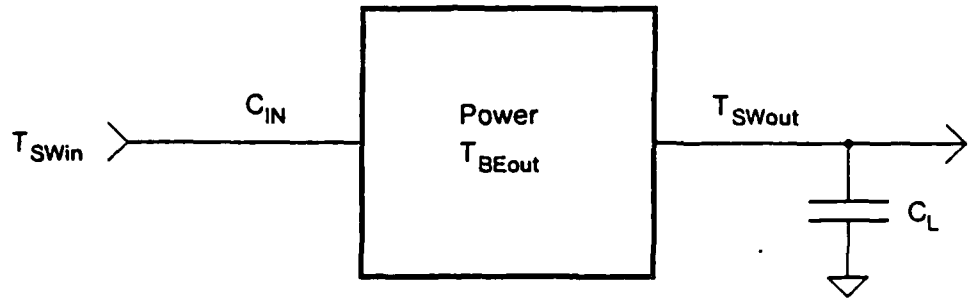


Figure 2: Macromodel Representation

3.3. Inverters

We begin our macromodeling analysis with the ubiquitous inverter, illustrated in Figure 3. The results will be extended to more general gates in a subsequent section. For the sake of conciseness, our analysis is only shown for rising input, falling output nMOS gates. The macromodel equations for the opposite transition and for CMOS are similar. We will present the actual macromodel equations (for both transitions) in the section on general logic gates.

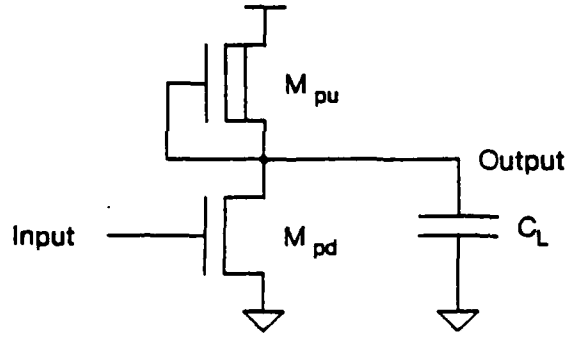


Figure 3: A Depletion Load nMOS Inverter

3.3.1. Objective Function

The practicing engineer typically must design circuits such that they satisfy delay specifications. The engineer also desires to minimize some objective function subject to those delay constraints. Power dissipation is a major concern in ratioed nMOS technology. We accordingly choose to minimize power dissipation, which for nMOS is dominated by static power consumption. The static power consumed by an nMOS inverter is roughly proportional to the

shape factor of the pullup; that is,

$$Power = a_1 + a_2 S_{pu}$$

where a_1 and a_2 are constants that depend on the fabrication process and power supply voltage.

The choice of an objective function for CMOS circuits is not as clear. Usually a designer wishes to minimize area, power dissipation, or some combination of the two. Characterizing the area consumption is difficult because it is highly dependent on layout style. However we can easily describe the contribution of the transistors. This is simply

$$Area = Poly\ Pitch \times \sum_{i=1}^{n} stack\ width_i$$

where we have omitted the transistor lengths because for CMOS they are set to the minimum channel length.

3.3.2. Output Waveform

Computational limitations mandate the use of a simple delay model. The simplest transistor representation that provides tolerable accuracy is a switched resistor. The MOS transistor is modeled with a capacitor from the gate to ground and a switched linear resistor from drain to source. The gate to source voltage controls whether the resistor is switched on or off. The delay characteristics of the model, along with their implications for circuit optimization, have been analyzed by Hoyte and Glasser in [8] and [9]. The principal advantage of the model is its simplicity, which allows one to derive closed form expressions for the optimal transistor sizes, leading to fast run times. Unfortunately the model can be alarmingly inaccurate. Moreover the errors can be exacerbated by the optimization. These workers found that for a chain of similar gates where the capacitive loading on each stage is dominated by the input capacitance of the next stage (rather than the wiring capacitance), pushing the chain for speed results in equal stage delays. That is, if for a given input transition we desire that the chain respond as quickly as possible, the model predicts that the total delay be uniformly apportioned among the gates. A gate with a rising output switches just as quickly as one whose output is falling. For nMOS this virtually guarantees that while stages with rising outputs are insensitive to input waveshape, those with falling outputs are highly sensitive to input slope. This sensitivity means that the pulldown transistor cannot be accurately modeled as a resistor, and the effect on total chain delay is significant because the stage delays are equal. Rising output stage delays, for which the resistive model tends to be valid, do not dominate the total delay. The model exhibits errors of up to 70%, clearly unacceptable for serious circuit design.

Faced with the inability of the resistive model to account for waveshape effects, we are compelled to derive a more elaborate model. Ever mindful of computation time limitations, we pursue the simplest possible extensions that will provide the needed accuracy. We begin by studying the inverter's response to different input waveform slopes, paying particular attention to the different regions of transistor operation.

As the inverter's input rises and its output falls, the pullup and pulldown transistors sequence through different regions of operation. These regions are summarized in Table 1. For

the fast input response[3] the bulk of the delay accrues from the last states where the pulldown is in its linear region. Hence the pulldown can be approximated by a resistor, and the resistive model works well here. However for slow inputs the pulldown is saturated for a significant portion of the transition, causing the inverter to behave like an amplifier. In this mode the inverter is highly sensitive to the input waveform and consequently the resistive model breaks down.

Fast Input Response		*Slow Input Response*	
pullup	*pulldown*	*pullup*	*pulldown*
linear	off	linear	off
linear	sat	linear	sat
linear	linear	sat	sat
sat	linear	sat	linear

Table 1: Pullup/Pulldown Regions of Operation

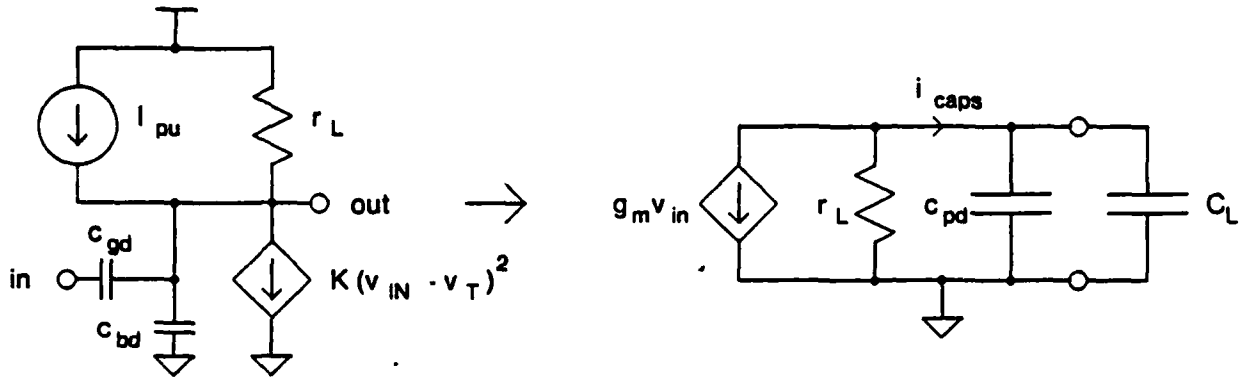
We seek a simple model that includes both amplifier and resistor behavior. We are especially concerned with the middle and latter parts of the input transition, for it is here that the inverter's output is sinking the most current. The beginning of the transition, where the noise margin requirement limits the output current, is not as crucial. For slow inputs the inverter can be modeled as an amplifier when the pulldown is saturated, and as a resistor tied to V_{OL} when the pulldown is in its linear region. As the input transition becomes faster, the inverter spends proportionately less time as an amplifier and more as a resistor. The mapping of the inverter to an amplifier and resistor is shown in Figure 4. We begin with the amplifier model when the input voltage reaches V_{IL}. This is an ac model; it measures perturbations from $(v_{IN}, v_{OUT}) = (V_{IL}, V_{OH})$. We change to the resistor model as the pulldown transistor firmly enters its linear region. For continuity of v_{OUT} and i_{OUT} when the model changes from an amplifier to a resistor, we use the resistor model when $V_{OL} \geq r_{pd}\, i_{caps} + v_{OUT}$.

We can acquire much insight into the behavior of the model by studying its drive curves [7]. These curves show the model's v_{OUT} versus v_{IN} relationship for different values of output load current. Figure 5 presents the circuit that measures the relationships; Figure 6 compares the model's drive curves with those of an actual nMOS inverter. In both of the drive curve figures, the inverter's DC transfer curve is indicated by the solid curve, and corresponds to $I_L = 0$. The transfer curve shifts as I_L becomes nonzero.

The model's drive curves possess several features that have important implications. First, the DC transfer curve is flat outside of the range $v_{IN} \in [V_{IL}, V_{IH}]$. This means that the gate will not respond until a rising input has reached V_{IL}, or until a falling input has dropped to V_{IH}. These "dead zones" provide immunity to noise in the input signal. Second, the two modes of model operation are clearly visible. The horizontal lines on the left and right sides of the figure display the resistive behavior. Here changes in output current produce a proportionate shift in output

[3] "Fast" means that the input transition time is fast relative to the output transition time.

Amplifier



Resistor

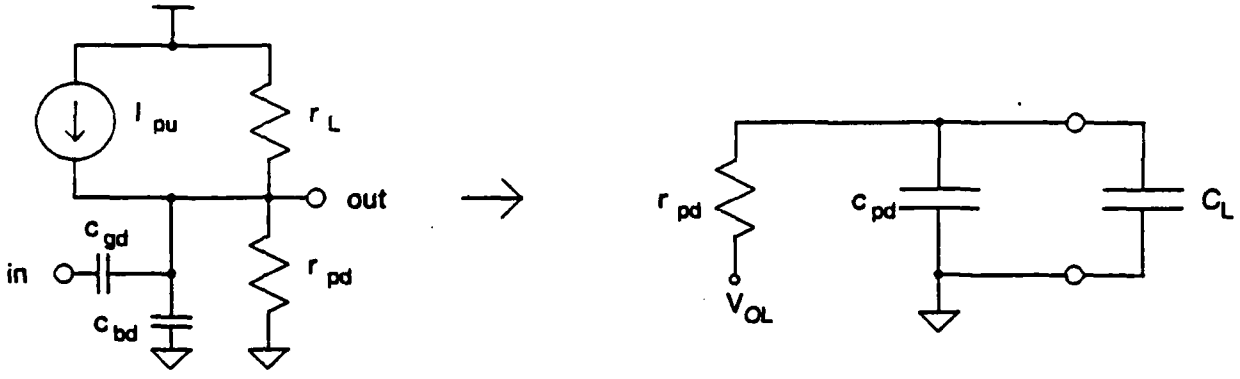


Figure 4: Delay Mapping

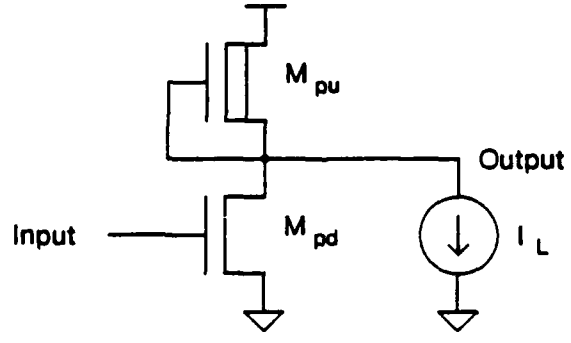
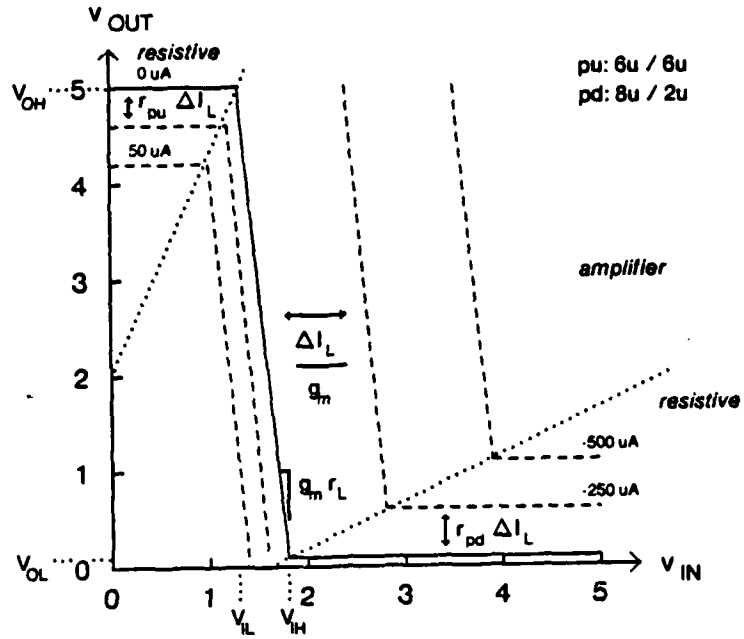
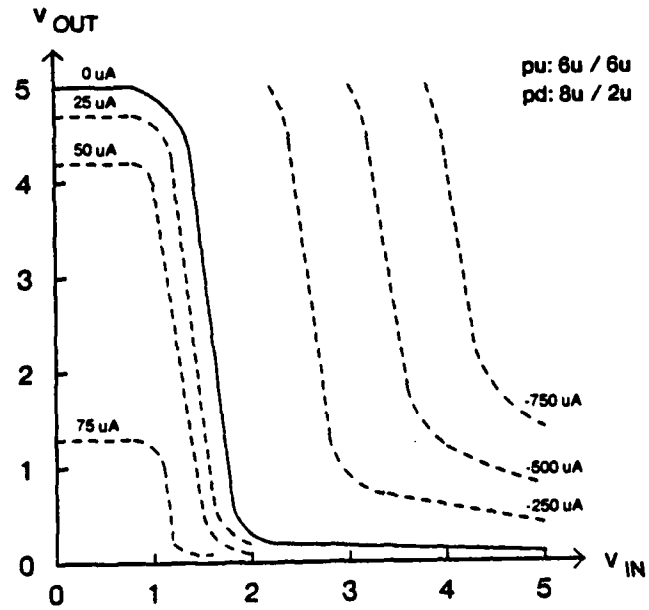


Figure 5: Circuit for Determining Drive Curves



Drive Curves for the Inverter Model



Drive Curves for an Actual Inverter

Figure 6: Comparison of Drive Curves

voltage,[4] and the output voltage is independent of input voltage. The amplifier mode is represented by the slanted lines in the center of the drive curves. Here the output waveform is entirely determined by the input waveform, the output slope being proportional to the input slope. For sufficiently slow inputs the inverter's output will closely track the DC transfer characteristic, exhibiting a strong dependence on the input waveform's shape. For faster inputs the inverter will not be able to track the input waveform, and will be forced out of the amplifier regime into the resistive regime. Therefore the inverter becomes less sensitive to input waveform shape as the input transition speeds up.

The ac model used for the amplifier behavior clashes with traditional engineering philosophy. Normally one creates an ac model by linearizing a circuit about a quiescent operating point. Here however we are interested in large signal behavior. Consequently, while we can perturb the system from an initial point, we have no easy method to calculate the model's parameters such as g_m and r_L. We cannot simply evaluate the parameters at a quiescent operating point because we have none. We instead view the problem at a more objective-oriented level, and seek to determine which values of g_m, r_L, etc., will provide the closest approximation to observed response times. Moreover, rather than using the same set of parameter values for the rising input and falling input responses (which would correspond to using a single group of drive curves to characterize the inverter), we procure additional accuracy by using distinct sets for each transition's begin and switch responses. This leads us to the following strategy: analytically derive expressions for the form of the macromodel equations, then curve fit to observed data to solve for the parameters in the equations.

Closed form expressions for the model's response to different input waveforms can be derived. Here we will outline the basic concepts; the actual equations will be given at the end of section 4. For very fast inputs the inverter changes from an amplifier to a resistive form immediately. In other words, the first order resistive model is valid. Figure 4 shows the model. For fast inputs the inverter model does not change immediately, but does change before the output transition completes. We use the amplifier model of Figure 4 but omit the resistor r_L. The output switches quickly enough that the current in the total capacitance C_{total} ($= C_L + c_{pd}$) dominates that of r_L, allowing us to neglect the resistor. For moderate inputs the input waveform is slow enough that the model changes after the output has fallen. The current in C_{total} still dominates that of r_L. For slow inputs, the current through r_L can no longer be neglected. Unfortunately this leads to equations which cannot be solved for closed form expressions for T_{BE} and T_{SW}. For very slow inputs, the input and output waveforms have slowed to the point where the current through C_{total} is almost negligible compared to that through r_L. The amplifier system reaches steady state, exhibiting a constant tracking error to the ramp input, being entirely limited by the speed of the input [10].

Combining the above analyses gives us the response for the entire range of input waveforms. This is illustrated in Figure 7. Figure 8 shows actual data for the rising and falling

---

[4]The reader may have noticed that this is not true for the low v_{IN}, high v_{OUT}, portion of the actual inverter's drive curves for larger I_L. This is because the pullup has moved from its linear region into saturation, and is behaving like a current source rather than a resistor. However the macromodel equations that we will develop depend only on the pullup's current sourcing ability being proportional to its shape factor, which will be true whether the pullup is best modeled as a resistor or as a current source. This is a consequence of our delay parameterization; T_{BE} and T_{SW} measure the duration of the inverter's response, which is inversely proportional to shape factor for either a resistor or current source model.

output transitions. The figures also include curves for T_{TPout}, the delay from when the input signal crosses the inverter's trigger point voltage to when the output does. Note the differences in the actual data for the two transitions. For nMOS technologies, static gates turn on both pullup and pulldown networks in order to generate a low output. Guaranteeing a valid output low requires that the resistance of the pulldown network be much lower than that of the pullup network. For very fast inputs we expect T_{SWout} to be roughly constant. This is apparent for the rising output transition, but for the falling transition, due to the low r_{pd}, this region is scarcely noticeable. The effect can also be seen in the T_{BEout} and T_{TPout} curves. The input to output coupling capacitance and the pullup/pulldown resistance form a high pass filter. Since r_{pu} is much larger than r_{pd}, the filter's cutoff point for rising outputs occurs at slower inputs than it does for falling outputs. Furthermore, because the pulldown transistor is in its linear region during fast falling input transitions, c_{gd} is larger, and more of the input transition couples to the output. Consequently the T_{BERout} and T_{TPRout} curves become convex as we move toward very fast inputs. The capacitive coupling is forestalling the rise of the output. The dissimilarity in resistances also implies a low trigger point voltage. For the same input slope, it takes longer for a falling input to transit the dead zone and approach the trigger point voltage than it does for a rising input. Correspondingly the T_{BERout} curve is steeper than the T_{BEFout} curve. For CMOS technologies the resistances are nearly symmetric, and both output transitions exhibit curves resembling those shown for the nMOS falling output transition.

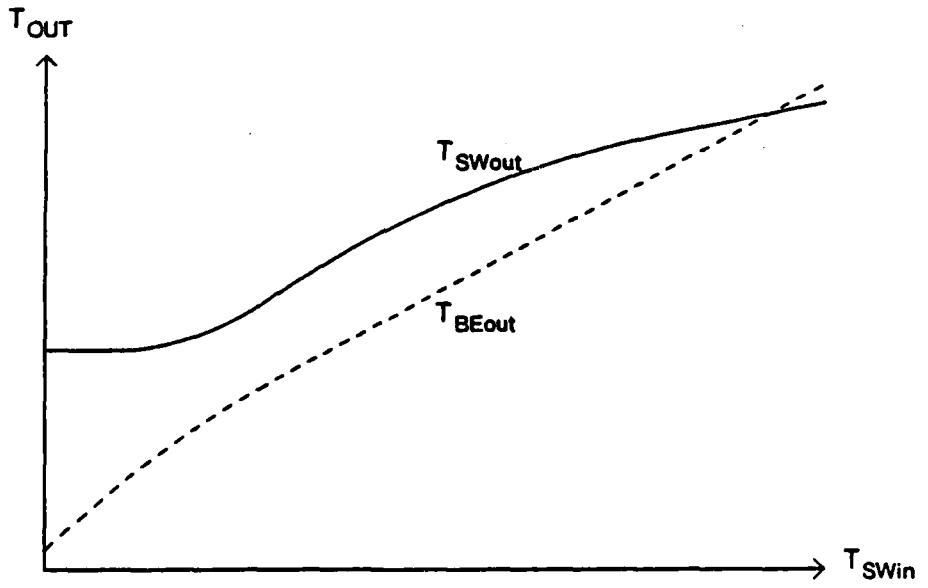
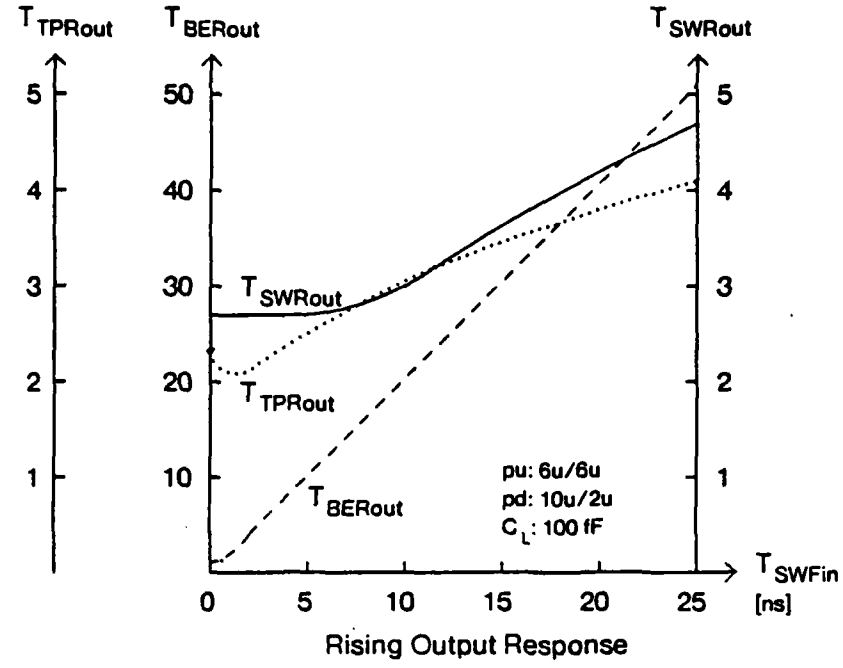


Figure 7: Inverter's Predicted Response

Having described a method for determining the inverter's response to various input slopes, we now seek a means of combining the results into one conglomerate expression. It is common to use smoothing functions to effect this combination. However many workers fail to consider the computational overhead incurred with these functions. We instead create simple functions that exhibit the desired behavior in each of the input slope regimes. To avoid placing any unnecessary burdens on the optimization algorithms, we choose functions that are twice continuously differentiable. Although optimization algorithms exist for solving problems with ill-



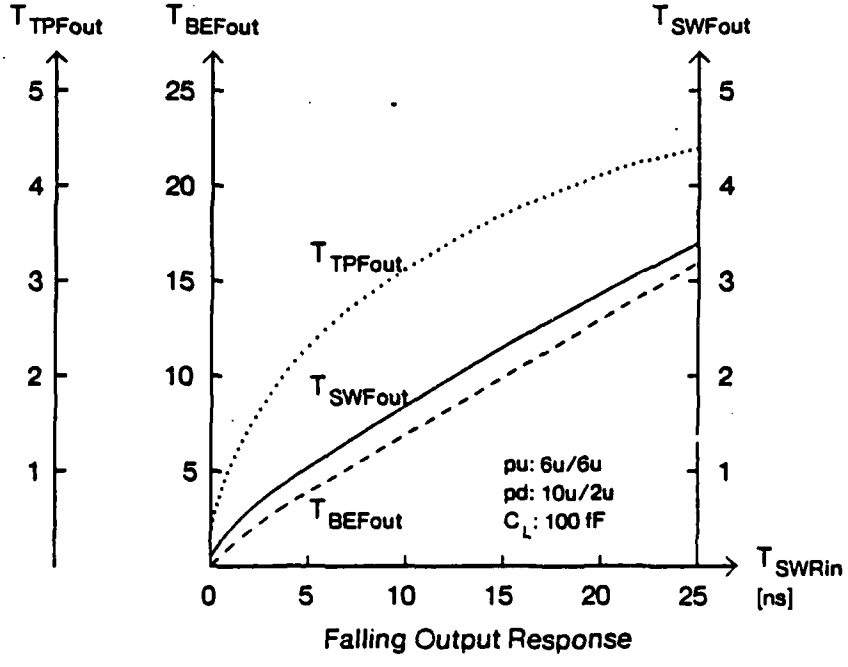


Figure 8: Inverter's Actual Output Responses

behaved (eg. discontinuous) functions, because of their added generality these algorithms tend to be slower. Moreover we prefer functions that do not contain multiple maxima or minima; ie. that are unimodal. This helps eliminate cusps that could trap an optimizer's iterative solution technique. The resulting inverter equations are fully described and analyzed in [11].

3.3.3. Input Capacitance

Calculating a gate's delay requires knowledge of the input capacitances of the gates that it drives. In this section we study an inverter's input capacitance. Our results will be extended to more general logic gates in a later section.

We begin by considering the components of the input capacitance. Figure 9 shows our model. The input capacitance has two constituents: the gate to drain and gate to source transistor capacitances.

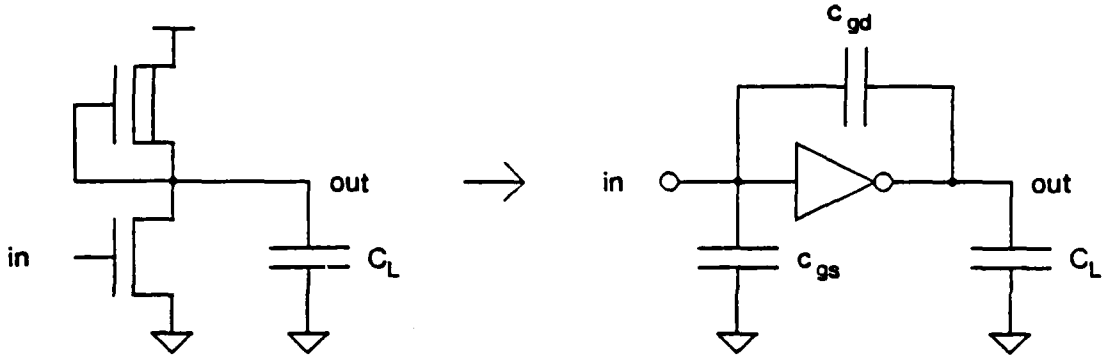


Figure 9: Input Capacitance Model

The input capacitance presented to the driver can change during the course of the input transition. This effect is largely due to the input to output coupling capacitance c_{gd}. Consider a rising input transition of moderate speed. During the beginning portion of the input waveform—that is, before the input voltage has reached V_{IL}—the inverter's output has not yet moved significantly. The input capacitance is therefore simply $c_{gs} + c_{gd}$. Both terms are proportional to the pulldown transistor's width.

As the input voltage passes V_{IL}, the inverter begins to pull its output low. Consequently the driver must supply more current to charge c_{gd} than it would have had the output voltage remained fixed. This is called the Miller effect [12]. The effective input capacitance has increased. Note that the total voltage change across c_{gs} is always $V_{OH} - V_{OL}$, while that across c_{gd} is 2 $(V_{OH} - V_{OL})$, but we are only interested in the capacitance seen during the beginning and switching portions of the input waveform. For very fast input transitions the output will not have moved until after the input has fully switched; hence the driver will not have seen any Miller capacitance during the actual transition. As we slow the speed of the input transitions, more of the output's switching time overlaps with the input's and we see more Miller capacitance. Eventually all of the output's switching time overlaps with the input's and C_{SWin} reaches a plateau. The expected behaviors of the effective input capacitances C_{BEin} and C_{SWin} appear in Figure 10.

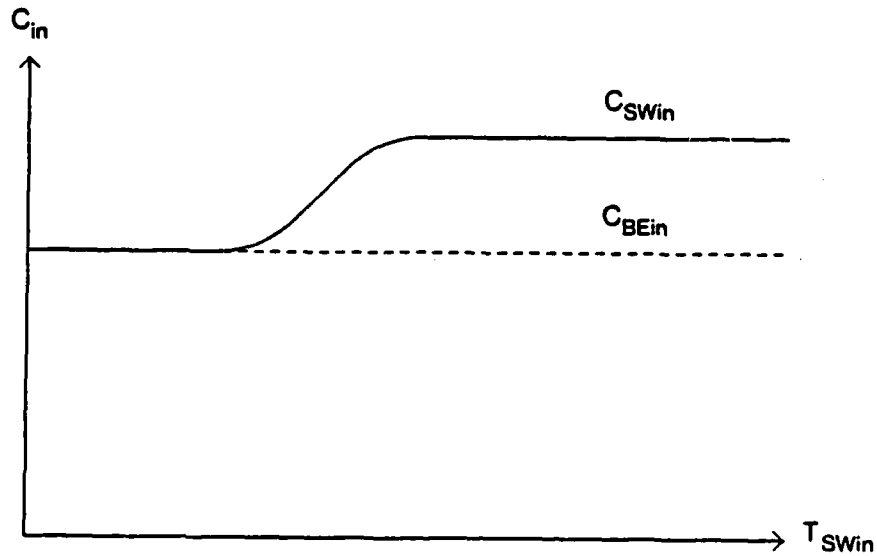


Figure 10: Expected Input Capacitance

The analysis is complicated slightly by the fact that since c_{gd} and c_{gs} are functions of v_{GS} and v_{GD}, they not only vary as the gate switches, but their average value during the input transition changes as the input transition slows down. The outcome is that we see two effects as the input transition slows down: more of the output switches during the input transition, and the average c_{gs} and c_{gd} seen during the input transition changes. For the falling input, rising output transition the pulldown begins in its linear region, passes through saturation, and terminates in cutoff. As the switching time of the input increases, the input's fall extends into the output's rise. This causes more overlap between the input and output transitions, but also raises the average v_{DS} during the input transition. The pulldown spends proportionately less time in its linear region and more in the saturation region. This implies that the average c_{gs} increases slightly while the average c_{gd} drops [13]. Hence, although the coupling capacitance c_{gd} has dropped, this is offset by the increased overlap between transitions, and the Miller capacitance is virtually unchanged. Figure 11 provides an illustration from SPICE simulations of an inverter. C_{SWFin} is roughly independent of T_{SWFin}, and only depends on the size of the pulldown.

For the rising input, falling output transition the pulldown traverses saturation and ends in its linear region. For fast inputs the pulldown will remain saturated for most of the input transition. As we slow down the input transition, more of the output's fall coincides with the input's rise, reducing the average v_{DS} during the input transition. Consequently the pulldown spends proportionately more time in the linear region and less in the saturation region. Hence c_{gs} diminishes slightly while c_{gd} increases. As we have seen, slower input transitions imply that more of the output transition overlaps with the input transition. Unlike the case for C_{SWFin}, here the two Miller capacitance effects have complemented rather than offset one another. The coupling capacitance and the input and output transition overlap both increase, giving rise to a significant Miller effect.

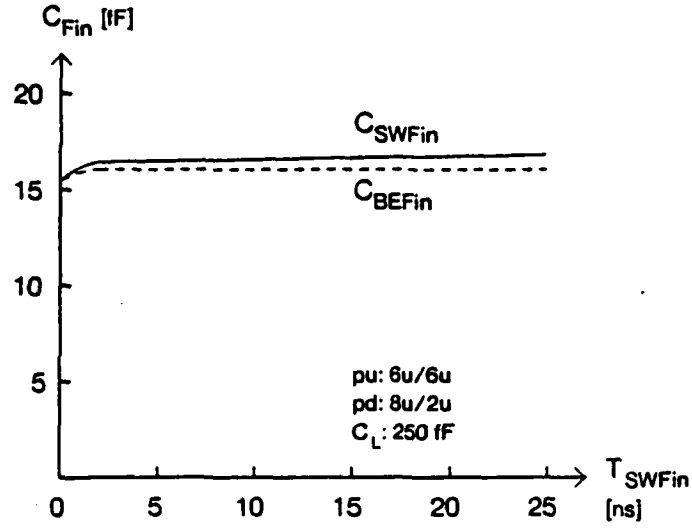


Figure 11: Capacitance for a Falling Input

Figure 12 displays data from SPICE simulations; the data agree fairly closely with our analysis. The input capacitance begins at a minimum, and then increases as the input transition slows, increasing the input to output transition overlap and the coupling capacitance. Once the input transition extends beyond the entire output transition, the overlap cannot be further increased and the curve reaches a plateau. Note that the interval of constant C_{SWRin} prior to the transition region is insignificant relative to the duration of the region. This arises because for very fast inputs the output begins to fall almost immediately. The drop in the pulldown's v_{DS} brings the pulldown out of saturation, causing it to exhibit significant c_{gd} during even fairly fast input transitions. Consequently C_{SWRin} begins to grow virtually as soon as the input transition slows and is no longer a step.

The slope of the transition region is governed by the load capacitance, transconductance, and c_{gd} of the pulldown. As the input transition slows the Miller effect increases. Since the coupling capacitance c_{gd} is proportional to the width of the pulldown, this increase in Miller effect, and hence in the curve's transition slope as well, must also be proportional to the width. The load capacitance and transconductance are pertinent because of their influence on the output waveform. For transistor sizes and loads that provide fast output waveforms, the transition region will be narrow because the input transition need only slow down slightly in order to completely overlap the output transition. This implies a steep slope. In contrast, large C_L and small g_m will lead to relatively shallow slopes.

The following expressions give our final equation for C_{SWRin}. Here T_{SWRin} is the switching time of the rising input waveform, C_{SWR0in} is the input capacitance for very fast input transitions, $C_{SWR\infty\ in}$ is the input capacitance for slow input transitions, and m is the slope of the curve during its transition from fast to slow input behavior. The macromodel curve fit routines solve for the parameters a_i, b_i, and c_i that give the closest correlation to SPICE simulations.

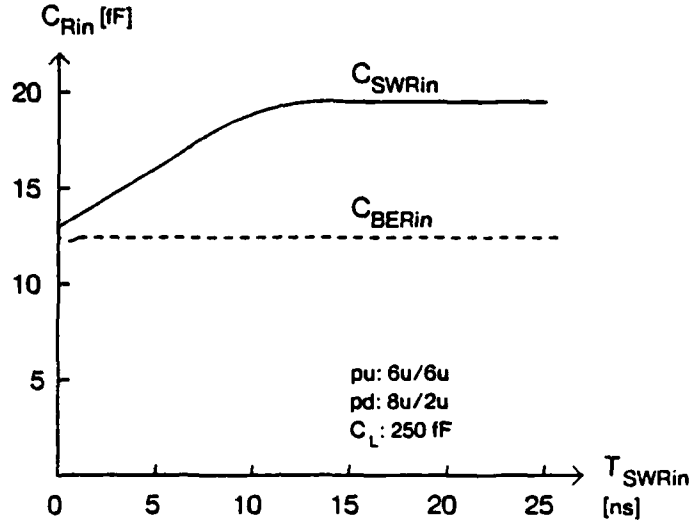


Figure 12: Capacitance for a Rising Input

$$C_{SWRin} = C_{SWRO} + \Delta C_{SWRin}\left(\frac{m T_{SWin}}{\Delta C_{SWRin} + m T_{SWin}}\right)$$

$$\Delta C_{SWRin} = C_{SWR\infty in} - C_{SWROin}$$

$$C_{SWROin} = c_{gs} + c_{gd} \approx a_1 w_{pd}$$

$$C_{SWR\infty in} = c_{gs} + 2c_{gd} \approx b_1 w_{pd}$$

$$m = w_{pd}\left(c_1 + c_2'\frac{g_m}{C_L}\right) \approx w_{pd}\left(c_1 + c_2\frac{w_{pd}}{C_L}\right)$$

For a CMOS inverter both the rising and falling input switching capacitance exhibit significant Miller capacitance. Recall that the magnitude of the Miller capacitance is affected by the input to output transition overlap and the change in average c_{gd} and c_{gs} as the input waveform slows down. As we saw in the nMOS case, for a transistor that is being turned off the two effects roughly cancel, while for a transistor being turned on they complement one another, producing a substantial Miller effect. Since for a CMOS inverter an input transition always turns on one device and turns off another, we always see a Miller effect capacitance, and model both C_{SWFin} and C_{SWRin} as above.

3.4. General Logic Gates

Inverters are but one of a myriad of logic gates found in circuit designs. In this section we will extend our discussion to cover a more general class of logic gates. We limit our analysis to logic gates with a single active input, as shown in Figure 13. Transitions at multiple inputs are not supported by our abstract model; accurate evaluation of their effects requires a low-level simulator that computes node voltages and branch currents. We feel that this represents an excessive computation cost and therefore choose a worst case gate state with a single active input to model multiple input transitions.

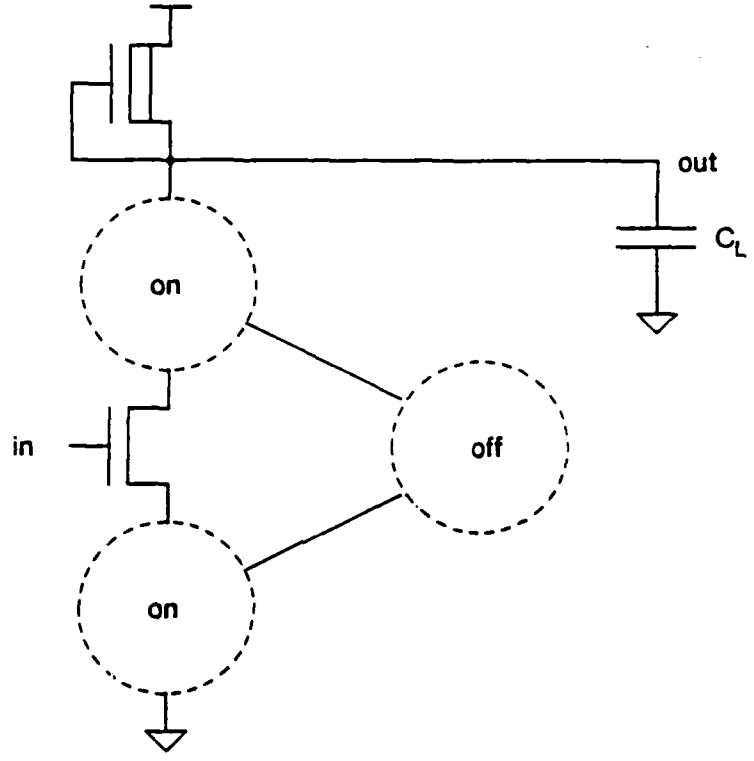


Figure 13: General Logic Gate

We will derive macromodel equations for the general logic gate by extending our inverter equations. As regards the objective function—be it power or area—the equations are basically unchanged. The power consumption of an nMOS gate is still proportional to the shape factor of the pullup, and the power or area consumption of a CMOS gate is still dependent on the stack widths. However the equations for the output waveform and input capacitance require moderate extensions.

3.4.1. Output Waveform

Additional transistors in a logic gate introduce two complications. If they are part of the path that switches the output by forming a path to V_{DD} or ground, their resistance and capacitance impede the output transition. If they are included in a side path that does not connect the output to a supply rail, their channel capacitance could add to the load capacitance and hinder the output transition. During the output switching transient, transistors with high

inputs are predominantly in their linear region. Hence we model them as RC lines formed of their drain to source resistance and channel to gate and substrate capacitance.

Figure 14 contains an example. Note that while the top transistor in the right pulldown branch is not in the conducting path, its capacitance adds to the total load. The general situation is depicted in Figure 15.

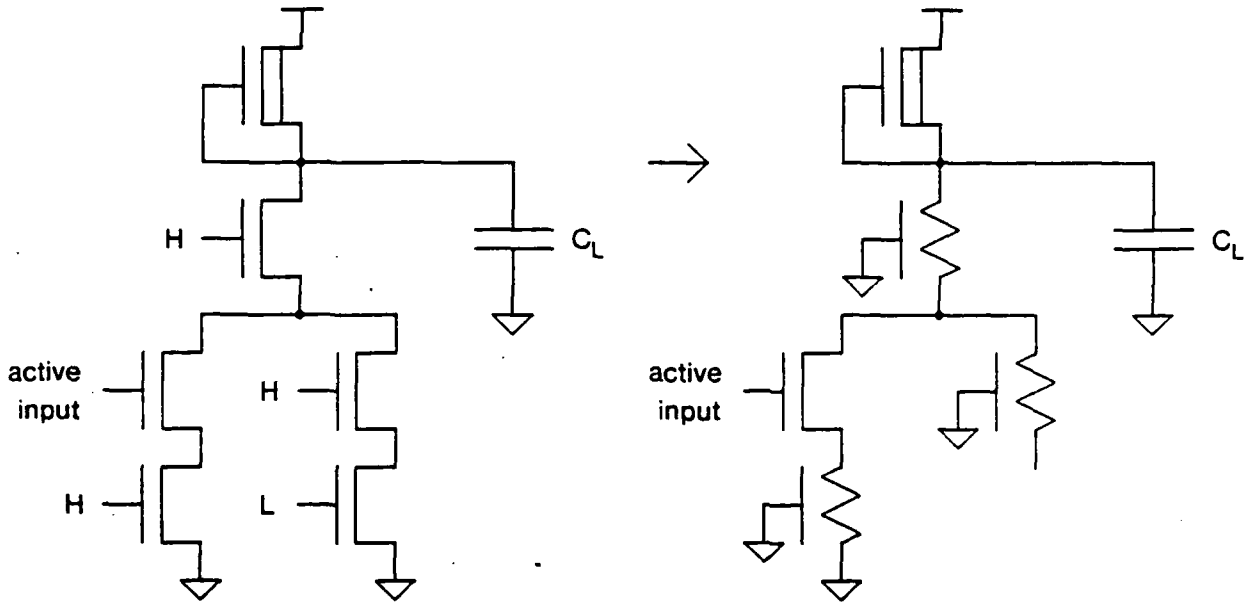


Figure 14: Example of a General Logic Gate

The additional transistors affect the gate's response in two ways. For fast inputs, the switching transistor can still be modeled as a resistor with its c_{gs} and c_{gd}, but a closed form expression for the output waveform cannot be derived. We instead find the approximate response by using an approach first proposed by Elmore [14] and now used in waveform estimation and bounding work in MOS circuits [15, 16]. This approximates the true response as a single time constant exponential. For slower inputs the speed of the output transition is limited by the slope of the input and transconductance of the switching transistor. Consequently, transistors in the conducting path which are electrically after the switching transistor have small v_{DS} and we can neglect their voltage drops. However we must add their capacitances (along with those of any transistors connected to them) to the total load capacitance. Transistors which are before the switching transistor do not impose any additional load since their capacitances are already discharged; nonetheless their resistance will decrease the switching transistor's effective g_m if they are in the conducting path, impeding the output transition. This effect is illustrated in Figure 16. The effective g_m has been reduced to $g_m / (1 + g_m r_b)$.

Combining our results, we obtain the following equations for the output response of the general nMOS logic gate. For each equation, the macromodeling support package finds the

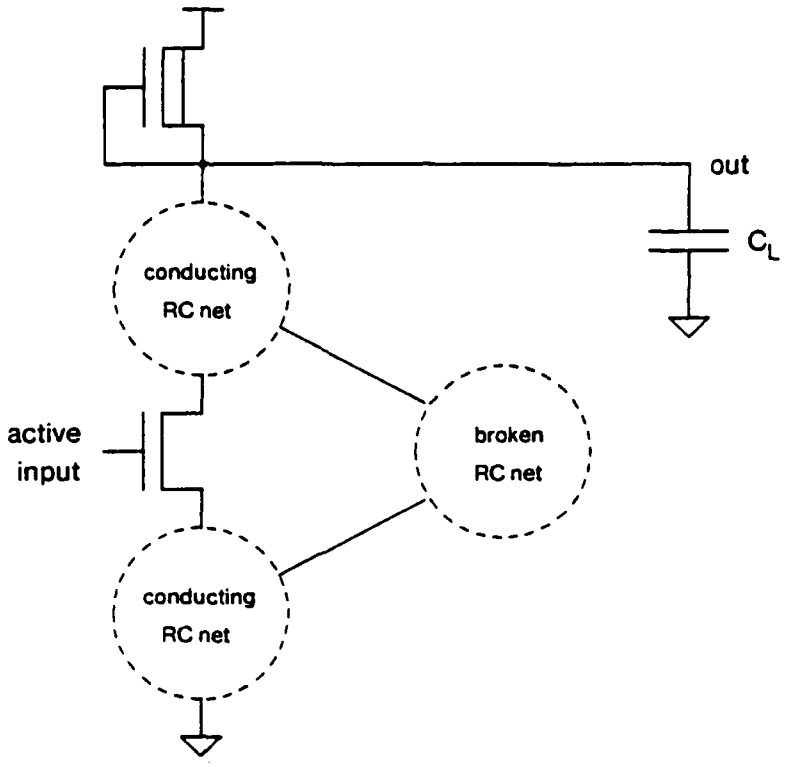


Figure 15: Circuit Model for a General Logic Gate

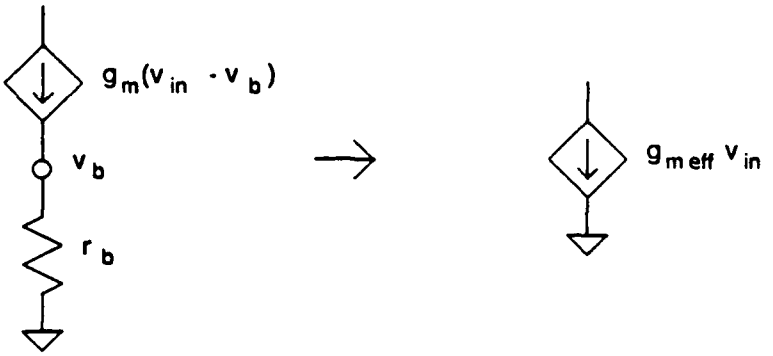


Figure 16: Reduction of Effective Transconductance

parameters a_i, b_i, etc., that provide the closest agreement with SPICE data.

Let w_{pd} = width of the switching device

$w_{pd\,total}$ = effective total w_{pd} of devices in the conducting path (treat as if w_{pd}'s were conductances)

$w_{pd\,before}$ = effective total w_{pd} of devices before the switching transistor in the signal path

$$C_{total} = c_{pd\,driver} + c_{pd\,after} + C_L$$

$$c_{pd\,driver} = c_1 w_{pd}$$

$$c_{pd\,after} = \sum_{after} c_2 w_{pd\,i}$$

Delay until output begins to fall

$$T_{BEFout} = T_{BEFO} + m T_{SWRin}$$

$$T_{BEFO} = a_1 + b_1 \frac{1}{w_{pd}} C_{total}$$

$$m = d_1 + d_2 \frac{S_{pu}}{w_{pd\,total}} + d_3 \frac{C_{total}}{w_{pd}}$$

Switching time of falling output transition

$$T_{SWFout} = T_{SWFO} + m T_{SWRin}$$

$T_{SWFO} = a_1 +$ Elmore delay approximation with $r_{pd} = \frac{b_1}{w_{pd}}$

for each conducting pulldown

$$m = d_1 + d_2 \frac{S_{pu}}{w_{pd}} + C_{total}\left(\frac{d_3}{w_{pd}} + \frac{d_4}{w_{pd\,before}}\right)$$

Delay until output begins to rise

$$T_{BERout} = T_{BERO} + m T_{SWFin}$$

$$T_{BERO} = a_1 + b_1 \frac{1}{S_{pu}} C_{total}$$

$$m = d_1 + d_2 \frac{S_{pu}}{w_{pd\,total}} + d_3 \frac{C_{total}}{w_{pd}}$$

Switching time of rising output transition

$$T_{SWRout} = \frac{(T_{SWRO} - T_{SWRO}^{asym})^2}{(T_{SWRO} - T_{SWRO}^{asym}) + m T_{SWFin} + (T_{SWRO}^{asym} + m T_{SWFin})}$$

$T_{SWRO} = a_1 +$ Elmore delay approximation with $r_{pu} = \frac{b_1}{S_{pu}}$

$$m = d_1 + d_2 \frac{S_{pu}}{w_{pd}} + C_{total}\left(\frac{d_3}{w_{pd}} + \frac{d_4}{w_{pd\,before}}\right)$$

$$T_{SWRO}^{asym} = e_1 + e_2 T_{SWRO}$$

3.4.2. Input Capacitance

As we have seen, the addition of extra transistors to form more complicated logic functions has an effect on a gate's output response. We have examined the effective capacitance of an nMOS inverter during the beginning and switching phases of the rising and falling input. Of these four modes, only one depended on the output waveform. The input capacitances during the beginning portion of the input transition had no dependence because the output was still stationary. The capacitance during the switching portion of a falling input had none because the average input to output coupling capacitance dropped as the input waveform slowed down, leading to no net Miller effect. Hence the input capacitance for these three modes depends only on the pulldown transistor's size, being proportional to the transistor's width (assuming a fixed channel length). Only the switching portion of the rising input possesses a significant output waveform dependence. To account for this dependence we must analyze the conducting path containing the switching transistor. Figure 17 shows an abstract gate model along with its circuit level representation. We model 'on' transistors as resistors (linear region approximation) and have added the appropriate capacitances from nonconducting paths to the total load capacitance.

We find that r_b causes a significant drop in the input capacitance. This fact has been exploited for many years by amplifier designers to raise input impedance and thereby improve the transfer characteristic. For very fast inputs, the contribution of c_{gd} is reduced by a factor $(1 + g_m r_b)$. For very slow inputs node *a* will have dropped to V_{OL} by the completion of the input transition; hence the input capacitance is identical to that of an inverter. For moderate inputs the input capacitance is in the transition region from fast to slow input behavior. Since the total capacitance increase during the transition region from fast to slow inputs grows, the slope of the transition increases.

Our analysis yields the following equation for CSWRin:

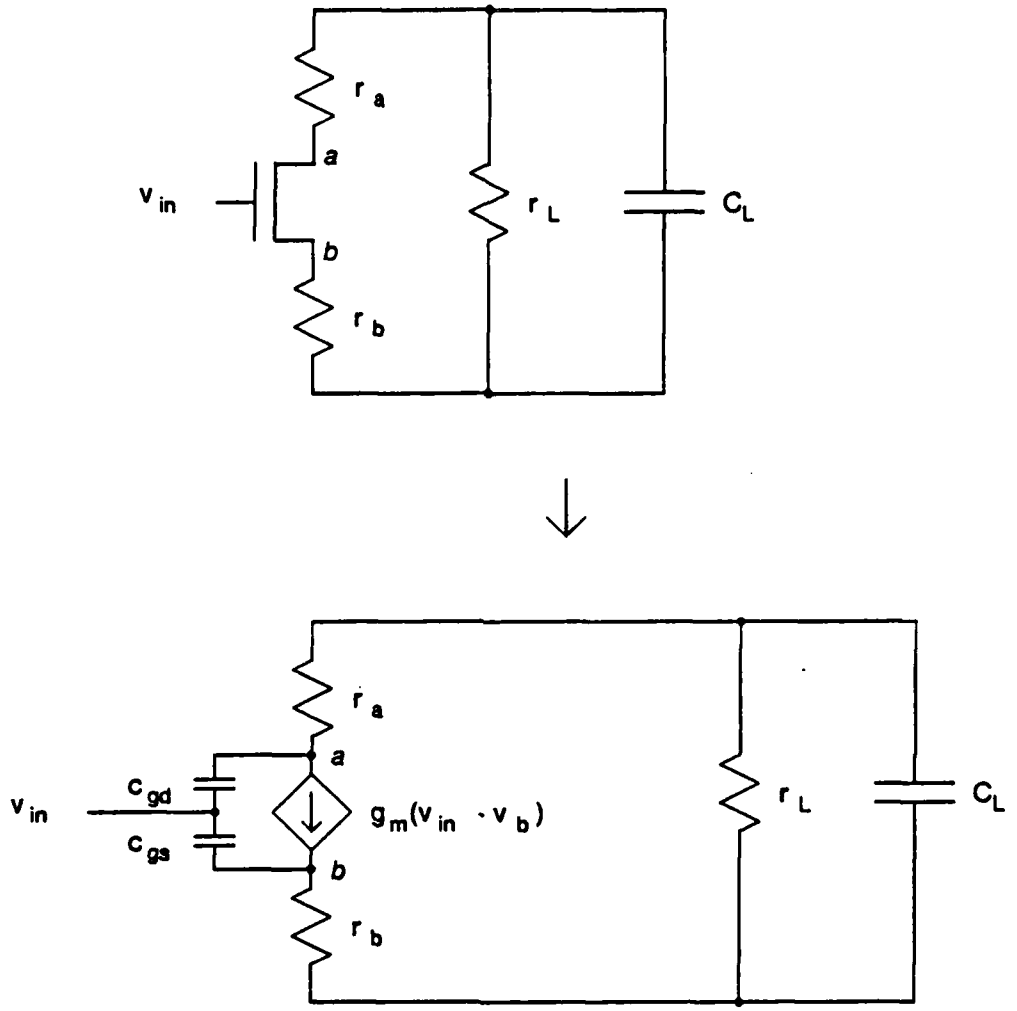


Figure 17: Circuit Model for Input Capacitance

$$C_{SWRin} = C_{SWROin} + \Delta C_{SWRin}\left(\frac{m T_{SWRin}}{\Delta C_{SWRin} + m T_{SWRin}}\right)$$

$$\Delta C_{SWRin} = C_{SWR\infty in} - C_{SWROin}$$

$$C_{SWROin} = \frac{a_1 w_{pd}}{1 + a_2 w_{pd} / w_{pdbcfore}}$$

$$C_{SWR\infty in} = b_1 w_{pd}$$

$$m = w_{pd}\left(c_1 + \frac{w_{pd}}{C_{total}}\left[c_2 + c_3 \frac{w_{pd}}{w_{pd\,before}}\right]\right)$$

Sample curves for a two-input NAND gate are shown in Figures 18 and 19. These curves illustrate the influence of the bottom pulldown transistor's resistance. In comparison with the capacitance curves for the bottom input, those for the top have a lower C_{SWR0in} and a steeper transition slope.

3.5. Implementation

We have developed a general purpose macromodeling software package. The modeler processes cell template files, a macromodel control file, and macromodel equations. Each cell template file contains a logic cell's general topology. The macromodeler inserts values for device sizes, capacitive loads, and input waveforms into the template, and then runs SPICE on the resulting circuit. The values of the input capacitance and output waveform are extracted from the SPICE output and stored. This process repeats for every combination of device sizes, loads, and input waveforms specified in the control file. At present 216 SPICE runs are performed for the general logic gate analyzed in the preceding section. The particular logic cells used are inverters and NAND gates. Owing to the simplicity of the cells, the SPICE simulations are quite fast, each requiring about ten cpu seconds on a DEC 20/60.

Once the data points have been obtained, the macromodeler solves for the parameters in the macromodel equations by using nonlinear curve fitting algorithms. We minimize the sum of squared error; minimizing the maximum error might also be acceptable but it is too sensitive to noise in the data. The curve fitter uses a Davidon-Fletcher-Powell algorithm [17] with modifications to accept upper and lower bounds on the parameters [18]. This is essential for ensuring that the final equations make physical sense. Otherwise local minima in the error function could draw the curve fitter toward nonphysical values for the parameters. Local minima in the error function also mandate that higher order effects be successively included in the model equations. That is, we solve for the first order terms in the equations first and then progressively solve for higher order terms. For example, when curve fitting the macromodel equation for output switching times, we first start with the simple RC model. We select a subset of data points with fast inputs and large capacitive loads — those points for which the model is most accurate — and solve for the RC terms in the equation. We lock these parameters and then solve for the waveshape terms. Next we solve for self-capacitance terms. Finally we unlock all parameters and curve fit again. This technique helps to guarantee that we reach the global minimum of the error function and adds very little to the total computation time because the time is dominated by the SPICE runs.

The modeler is written in a computer language called CLU [19]. It consists of SPICE interface, minimization, and matrix manipulation program modules. These modules contain 3200, 1800, and 1000 lines of code, respectively. All told, the modeling support routines comprise about 6000 lines of CLU code; the modeling programs specific to the general logic gate discussed in the previous section represent an additional 1700 code lines.

Pertinent curve fit statistics are shown in Table 2. The macromodel equations are typically

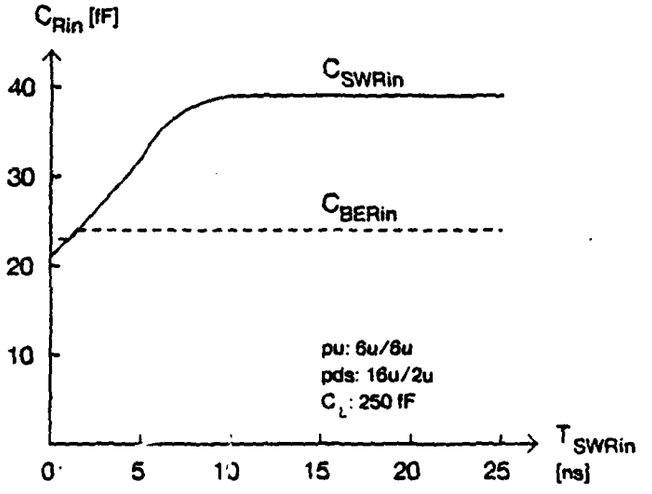


Figure 18: Input Capacitance for the Top Input of a NAND Gate

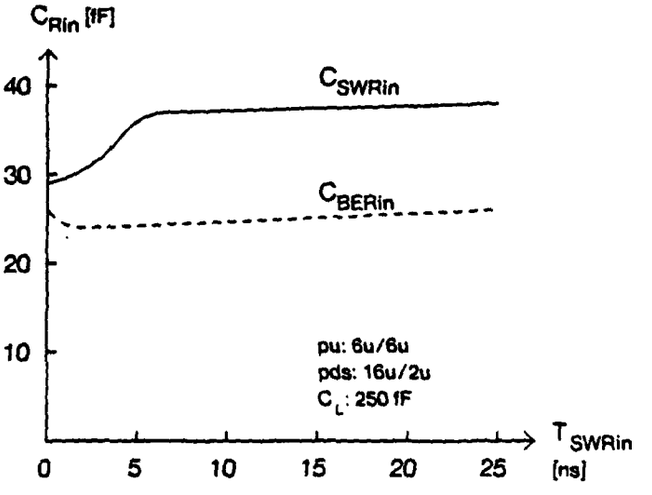


Figure 19: Input Capacitance for the Bottom Input of a NAND Gate

within several percent of the SPICE predictions, a major improvement over RC models. These benefits come at a small price in computational overhead because we have modeled the response of the entire cell, rather than using a more sophisticated transistor model and then having to compute the transistors' interactions to obtain the cell's response. The accuracy and computational speed of the macromodels make them well suited for both simulation and optimization applications.

Rising Input, Falling Output			*Falling Input, Rising Output*		
Model Eqn	% Error ave	max	Model Eqn	% Error ave	max
C_{BERin}	1.5	5.6	C_{BEFin}	1.5	6.9
C_{SWRin}	3.7	12.3	C_{SWFin}	1.3	9.7
T_{BEFout}	5.7	18.3	T_{BERout}	4.6	13.2
T_{SWFout}	8.6	27.8	T_{SWRout}	3.0	10.6

Table 2: Macromodel Curve Fit Accuracies

4. Optimization

4.1. Introduction

Having described the underlying logic gate models used by our CAD tool, we will now address the theory and implementation of our optimization algorithms. Our emphasis is on developing basic understanding and intuition rather than a rigorous mathematical structure via formal proofs. The theory of nonlinear optimization is fairly extensive and we cannot discuss all of it here; readers interested in more detailed treatments are referred to [20], [21], [22], and [2]. Additional information concerning our implementation of the algorithms can be found in [11].

Section 2 outlines the special features of the optimization problem, describing the properties of the objective and constraint functions. Section 3 presents the theory of the optimization algorithms. We select a method particularly suited to our problem, taking advantage of the properties of the digital MOS domain, and of our ability to create a circuit data base customized for the transistor sizing problem. Our approach, called duality, allows us to partition the problem into many simpler, smaller subproblems, and to transform the nonlinear delay constraints into a much simpler form. Section 4 discusses the implementation of the optimizer. We describe the organization of the software and study the optimizer's performance on some example circuits.

4.2. Properties of Our Problem

We begin by choosing an optimization technique that is appropriate for our problem. Selection of the technique is highly problem dependent, as "appropriateness" in nonlinear optimization is nearly synonymous with fast computation times, requiring that the optimization technique be closely matched with the problem's characteristics. We therefore commence by considering the properties of our optimization problem. We shall see that the problem can be separated, allowing us to apply a divide and conquer strategy and thereby greatly increase

computational speed.

We desire to minimize a circuit's power consumption subject to constraints on signal path delays and transistor sizes. The objective function, total power, is the sum of the power consumptions of each circuit cell. For nMOS the power consumption of each cell is solely determined by, and is linear in, the shape factor of the pullup transistor. For CMOS the power consumption is linear in the capacitive loads that must be driven, which are due to the area of the transistor gates and interconnect capacitance, but also depends somewhat on the input waveforms. Hence for nMOS, and nearly for CMOS, the power consumption of a circuit is a separable function of the form

$$P_{total} = \sum_{i=1}^{n} P_i$$

where P_i is a function of cell i only.

The problem's constraints are of two varieties: delay specifications and transistor size design rules. The delay along a signal path is a nonlinear function of the circuit's transistor sizes, and is nonlocal, being composed of contributions from each cell along the signal path. These contributions are not entirely independent, but fortunately transistor sizing is very nearly a separable operation, because both waveshape and capacitive loading effects diminish rapidly with electrical distance. Consider an inverter chain. Whether the input signal is slow or fast, by the time the waveform has propagated to the chain's output its shape will be predominantly determined by the last gate in the chain. Fast inputs put the gate in an RC response mode where the output waveform's switching time is governed by the gate's effective output resistance and capacitive load. Slow inputs place the gate in a gain limited mode where the gate's gain increases the sharpness of the waveform's transition. Thus a gate behaves as a crude wave shaper.

Capacitive loading effects also attenuate quickly with electrical distance. Suppose the chain is driving a large capacitive load. The last gate will have to be fairly wide in order to drive the load. The second to last gate will in turn have to be somewhat large to drive the wide pulldown transistor of the last gate. We need progressively less widening as we work our way backwards from the load. Within a few gates we reach a point where we are fully shielded from the size of the load.

Design rules and layout considerations restrict the minimum and maximum sizes that a transistor can have, and for nMOS there is also a minimum beta ratio (ratio of pulldown to pullup shape factors) requirement. The former is a box constraint; the latter is a linear constraint. The constraints on a circuit's transistors are entirely local to each cell, and are therefore separable.

Accuracy requirements are also important, and exhibit a peculiar ambivalence in our problem. The delay specifications on the signal paths must be met to the full accuracy afforded by the macromodels. However the power minimization is less critical. We can tolerate a small errors in minimizing the circuit's power consumption, especially if the inaccuracies are accompanied by large savings in computation time. In fact, at present designers use only crude heuristics or mostly ignore the power consumption issue.

In summary, the problem embraces characteristics ranging from the trivial to the extremely

difficult. The objective function is a simple summation of contributions from each logic cell, each contribution being linear in the cell's transistor sizes. On the other hand, we anticipate hardship with the delay constraints, since they are global and nonlinear in the circuit's transistor sizes. Fortunately there are not many of them; typically a designer will specify delays for only a few percent of the paths through a functional block. In contrast, the transistor size constraints are quite simple, consisting of linear and box constraints. However there are a large number of them, at least one for every transistor in the circuit, carrying the potential for huge run times. The objective and constraint functions are essentially separable, linearly composed of nearly independent contributions from the circuit's cells. We would prefer a nonlinear optimization algorithm that can exploit this separability, pursuing a divide and conquer strategy where the problem is partitioned into many smaller subproblems. This segmentation is beneficial because with most optimization algorithms, run times grow superlinearly with the number of design variables. Thus by breaking up a large problem, faster run times can be achieved. In particular, if the problem could be partitioned down to the cell level, the size of the vector space for each subproblem would be the number of transistors in each cell. Small vector spaces usually imply fast run times.

4.3. Duality

We carefully studied several optimization techniques, including feasible directions and penalty methods. Neither of these methods is capable of exploiting the near separability of the problem. Since we felt that partitioning was vital to achieving fast run times, we chose a technique called duality [20], a fairly exotic approach in comparison to the other two methods. We shall see that the computational efficiency that it affords more than compensates for its conceptual complexity. The basic idea of duality is to form a so-called dual problem which can be significantly easier to solve than the original, or primal, problem. In our case the primal is difficult to solve because of the global, nonlinear delay constraints and large number of transistors to size.

Duality offers several major advantages. First, the primal problem need not be feasible. This is a strong possibility because high-performance designs often push circuit topologies to the limits of their performance. It is likely that a designer will specify delays for some signal paths which cannot be met. In this event we desire that our CAD tool do its best to meet those speed specifications while optimizing the power consumption of the other paths whose delay constraints can be met. Duality achieves this goal. Second, inactive constraints pose no difficulty for duality. A designer specifies maximum delays along signal paths. Due to paths sharing common portions, it is possible that one path's delay specification will be exactly met while a companion path will be faster than required, and yet this situation minimizes power consumption. This is essentially a recasting of the critical path problem; the first path is one of the circuit's critical paths. Third, and perhaps most importantly, duality can be extremely efficient computationally. This is due to two factors. Duality converts the nonlinear delay constraints into simple box constraints, allowing us to apply fairly simple optimization algorithms (which implies robustness as well) with quasi-Newton methods. The quasi-Newton methods lead to fast convergence. Also, the dual approach permits us to exploit the separability of the power and delay functions, enabling us to use a divide and conquer strategy where each cell is optimized separately. Partitioning affords significant computation speed advantages.

Like any nonlinear optimization approach, the advantages are balanced by drawbacks. Duality is not applicable to all problems; it works best for those satisfying a certain convexity requirement, a property which digital MOS circuits possess. Another drawback is due to our partitioning approach rather than duality itself. Although exploiting separability provides run time

improvements, it necessitates the maintenance of additional data in the circuit's data base, along with a close interaction between the control structure and the data base. Partitioning the circuit into cells implies incremental optimization of each cell in succession. This mandates a sophisticated data base, and places profound requirements on the programming language chosen to implement the optimizer.

4.3.1. Lagrange Multipliers

Lagrange multipliers are the key to understanding duality. We shall explain their use and significance through a simple example. Consider a chain of two inverters. The input is driven by a source v_{IN} through a resistor R_S; the output connects to a load capacitance C_L. We wish to constrain the maximum delay of the chain. If we fix the width of the pullup transistor, the length of the pulldown, and the beta ratio of the inverters, then we can treat the power consumptions of the inverters as the only free variables because specifying the power consumption of either logic gate determines the gate's transistor sizes. Let P_1 be the power consumed by the first gate and P_2 be that consumed by the second, and suppose we desire the total delay $T_{total} = T_{IN} + T_1 + T_2$ to be less than or equal to some T^*.

This maximum delay specification places restrictions on the allowable power consumptions of the gates. Certain regions in (P_1, P_2) space will not meet the speed specification. For instance if the shape factors of the transistors in the second inverter are too small, the inverter will not be able to charge the capacitor C_L quickly enough to satisfy the delay constraint. On the other hand, if the shape factors are too large, implying a wide pulldown and hence a large input capacitance, the first inverter will not be able to drive the second quickly enough. Of course the first inverter's shape factors can be made larger to drive the extra load, but after a certain point the first inverter's input capacitance becomes so large that the delay through R_S precludes meeting the delay specification. Since power consumption is linearly related to shape factor, the bounds on the shape factors imply bounds on the power consumption. Similar reasoning applies to the power consumption of the first inverter, giving us the forbidden zones (dashed lines) shown in Figure 20.

We can more precisely characterize the feasible set of power consumptions. We do this by employing our macromodel equations for the inverters, allowing us to derive an analytic expression for the delay through the gates as a function of their transistor sizes. The resulting constraint surface $T_{total} = T_{IN} + T_1 + T_2 = T^*$ is elliptical as depicted in the figure.

Figure 20 also shows the correlation between total power and path delay. The dotted lines are contours of constant power. To meet the delay constraint we must stay within the elliptical region, but the total power dissipation varies with position in the region. As we move toward the upper right of the feasible set, the power dissipation increases. At the point *Max* we have reached the maximum power consumption that will still allow us to satisfy the delay constraint. Here the delay and power contours are tangent, and their gradients point in the same direction. If we instead work our way toward the lower left of the feasible set, the total power dissipation decreases. When we reach the point *Min* the dissipation will be at its lowest level that will still satisfy the delay constraint. Here the delay and power contours are again tangent, but now their gradients point in opposite directions; mathematically this can be expressed as

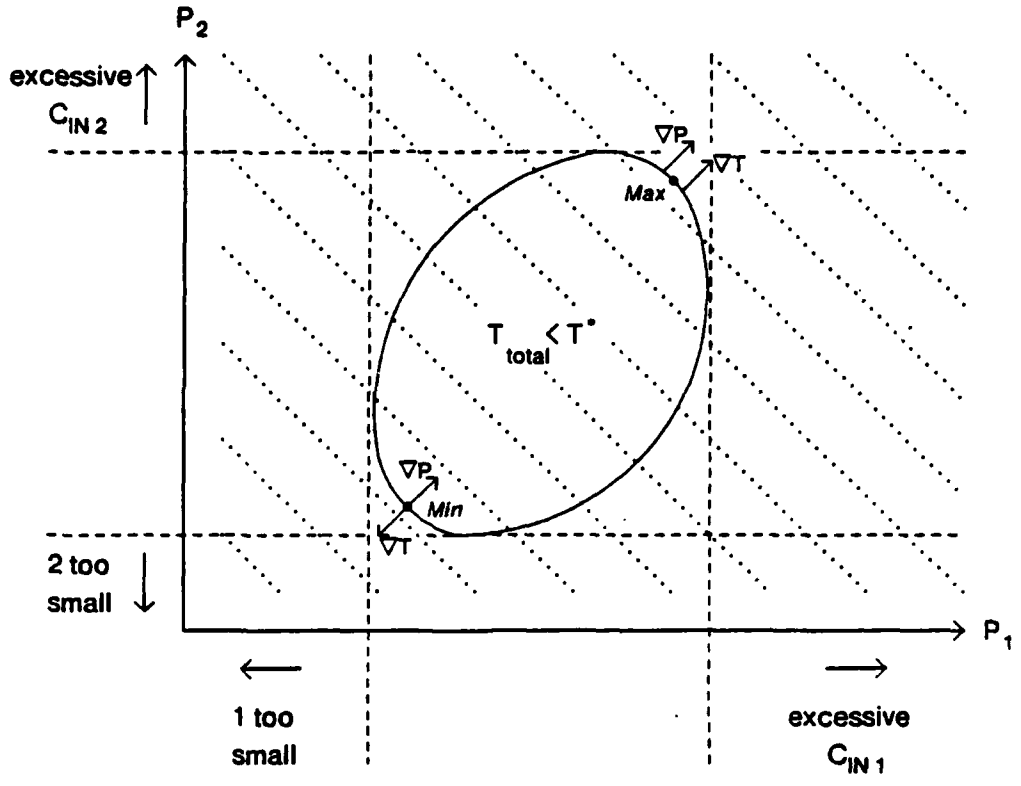


Figure 20: Contours of Delay and Power

$$\nabla P = -\mu \nabla T$$
$$\text{or } \nabla (P + \mu T) = 0$$
$$\text{where } \mu > 0$$

The variable μ is called a Lagrange multiplier, and offers the key to solving our nonlinear optimization problem.

4.3.2. Finding the Optimum

We can acquire an understanding of how to find the optimum by applying a graphical approach to the inverter chain. We are interested in the possible total power and total delay combinations that the circuit can exhibit. In other words, we desire the locus of points (T_{total}, P_{total}) that will be generated if we substitute all valid transistor size combinations into the circuit. This locus of points is denoted the set of all possible pairs, $\mathcal{P}$, and is displayed in Figure 21. The set's lower left boundary is the classic power-delay tradeoff curve (bold line); it represents designs that offer propagation delays with the lowest possible power consumption for those delays. Points toward the left of the curve are in the high-speed, high-power region. As we move down the curve to the right, we trade speed for reduced power consumption, and eventually enter the low-power, low-speed region.

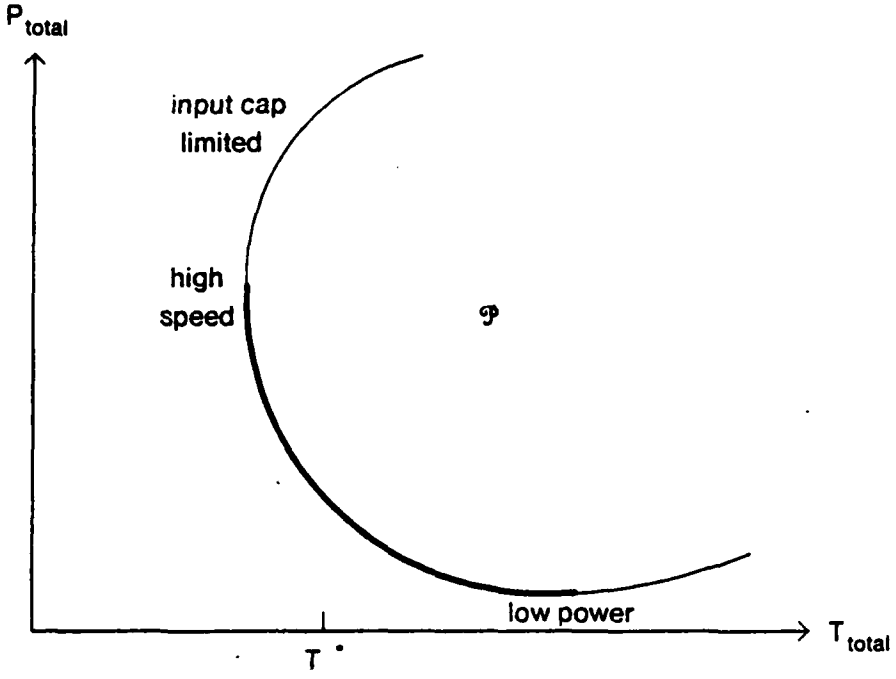


Figure 21: Set of Possible Pairs

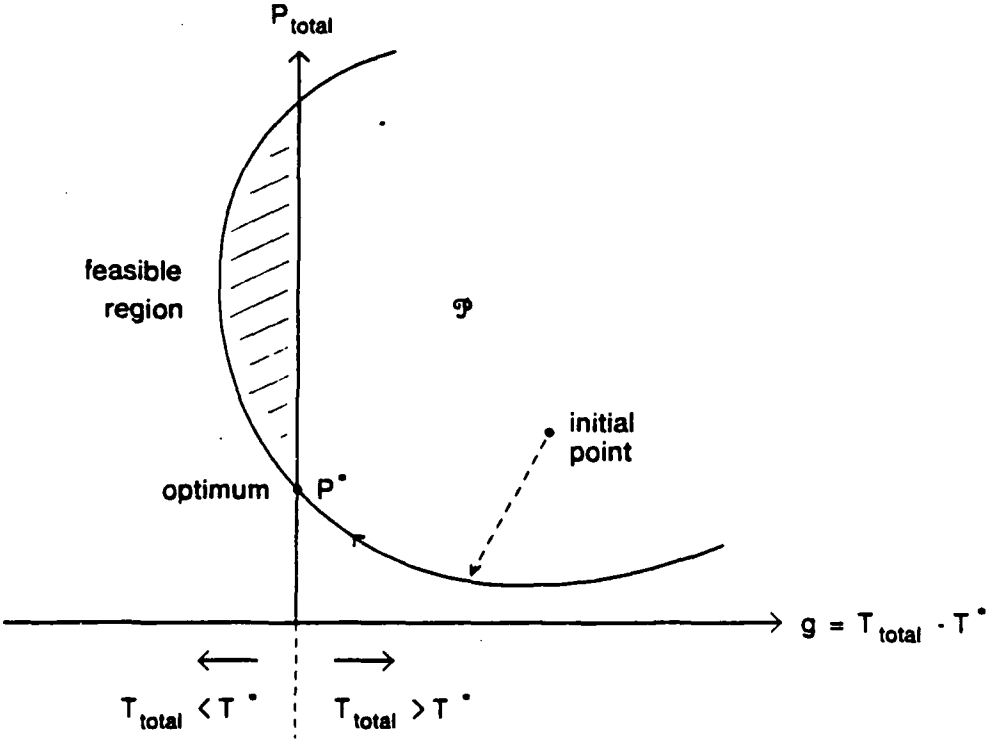


Figure 22: Reaching the Optimum

Points that are not on the tradeoff curve correspond to nonoptimal circuits. These circuits either consume more power than an optimal circuit with the same delay, or are slower than an optimal circuit with the same power consumption. For example, suppose the inverter chain is driving a large capacitive load. We should make the second inverter's shape factors relatively large in order to drive the load, and then make the first inverter slightly large to drive the wide pulldown of the second inverter. If we reverse the ordering, making the first inverter very large rather than the second, the circuit will still consume the same amount of power as the optimal one, but will be considerably slower.

Our delay specification restricts the points that we can accept to those having a total delay less than or equal to T^*. We can focus our attention on this subset by shifting the vertical axis as shown in Figure 22. Points to the left of the axis have delays which are faster than T^*; this subset is called the feasible region. The optimum is the point in this region with the lowest power consumption, and is located at $(0, P^*)$ in the figure.

We must somehow reach this optimum point starting from an arbitrary point in the set $\mathcal{P}$. The approach duality takes can be thought as a two-step process, illustrated in Figure 22. The first step is to move to and remain on the lower boundary of $\mathcal{P}$. The second is to walk along this boundary to the optimum. Note that while conceptually this process may be interpreted as two steps, it must be implemented as an inner loop embedded in an outer loop. Step one corresponds to the inner loop, and step two to the outer. This forces the search to follow along the lower boundary of the set.

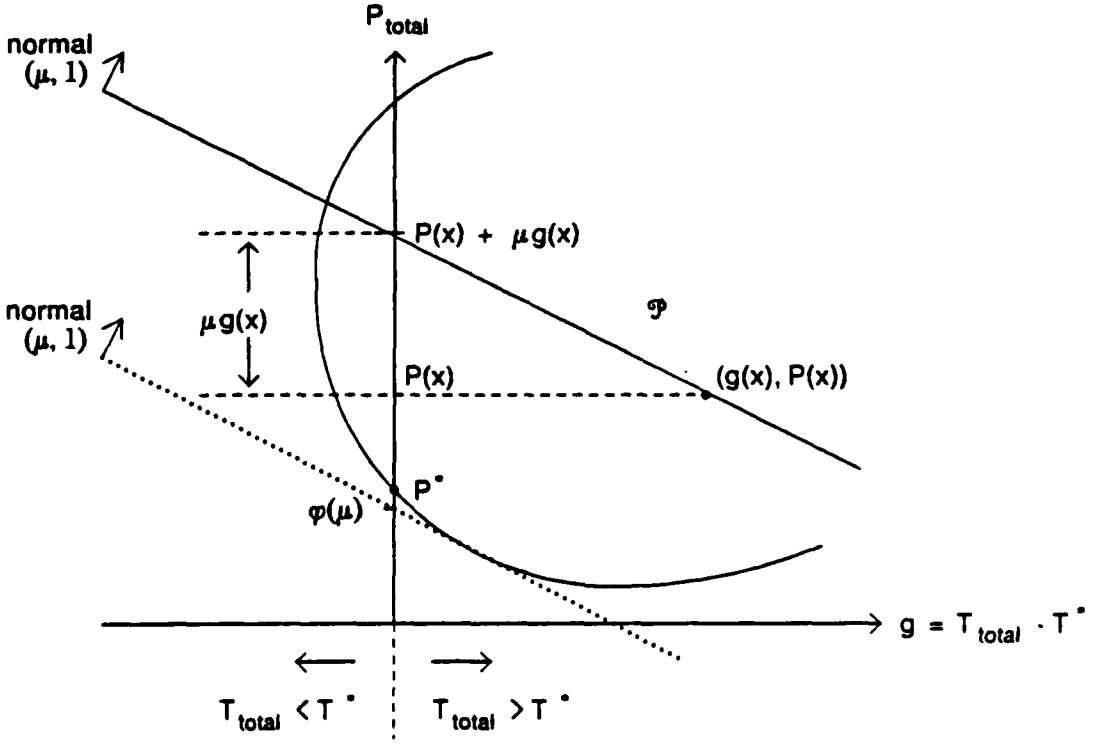


Figure 23: Inner Loop Minimization

We will now describe the implementation of each loop. Figure 23 gives a graphical representation of the inner loop. Suppose that we begin at some arbitrary assignment x of transistor sizes, with some arbitrary nonnegative Lagrange multiplier vector μ. The transistor sizes x map to point (g(x), P(x)) in g-P space. We can move from this point to the lower boundary of the achievable set by sliding the solid line down until it is tangent to the bottom of $\mathcal{P}$, while preserving the slope of the line. By geometry we know that a line through a point (g(x), P(x)) with normal (μ, 1) intersects the vertical axis at P(x) + μg(x). The multiplier μ fixes the slope of the line. Hence this sliding operation is equivalent to bringing the vertical intercept down while holding μ fixed. We must perform the minimization

min $\{P(\mathbf{x}) + \mu\, g(\mathbf{x})\}$
subject to $\mathbf{x} \in \mathcal{S}$, the set of valid transistor sizes,
with μ fixed

We shall denote the argument of the minimization as L(x, μ), the Lagrangian, and the minimum intercept as $\varphi(\mu)$, the dual functional. Note that since the new circuit produced by this minimization maps to the lower left boundary of $\mathcal{P}$, the circuit is well designed in the sense that it consumes the minimum power for the speed that it offers. Furthermore, since the minimization finds a point where the Lagrangian's gradient with respect to x is zero, we have $\nabla_{\mathbf{x}}[P(\mathbf{x}) + \mu\, g(\mathbf{x})] = 0$. This is equivalent to the optimality condition we derived in the preceding section, indicating that the circuit's power and delay gradients point in opposite directions.

Since the circuit's power and delay functions are composed of nearly independent contributions from each logic cell, we can separate the minimization, minimizing the Lagrangian of each cell in succession. Interactions among cells are propagated by a relaxation technique: we iterate the minimization of the entire circuit three times. Thus the minimization is partitioned over many small vector spaces, leading to fast computation times.

While this technique provides a tremendous savings in computation time, it does place rather elaborate demands on the data structures. When the minimizer sizes a cell, it is essentially performing an incremental optimization of the circuit. This is more involved than an incremental simulation. Information regarding boundary conditions (e.g., input waveforms and capacitive loading) imposed on a cell by its drivers and receivers must be maintained in the data base. It is this sophisticated data base with its closely coupled interaction with the control structure that makes the approach used by general purpose optimization tools inappropriate for our problem. These programs "black box" the circuit, interacting with it solely via a simulator. Hence the tools are in a sense isolated from the data base. They can neither access the circuit's connectivity description to guide partitioning nor embed additional information in the data base to assist the optimization. Their methodology does not support our incremental optimization approach with its accompanying savings in cpu time.

The outer loop walks along the lower boundary toward the optimum. We can gain insight into how this might be accomplished by contemplating the effect of different Lagrange multipliers on the inner loop's minimization. Figure 24 provides an illustration. We see that as we move toward the optimum point (0, P^*) the intercepts $\varphi(\mu_i)$ increase in value until they reach P^*. Conversely, if we move away from the optimum in either direction, the intercepts $\varphi(\mu_i)$ decrease. This is a maximization:

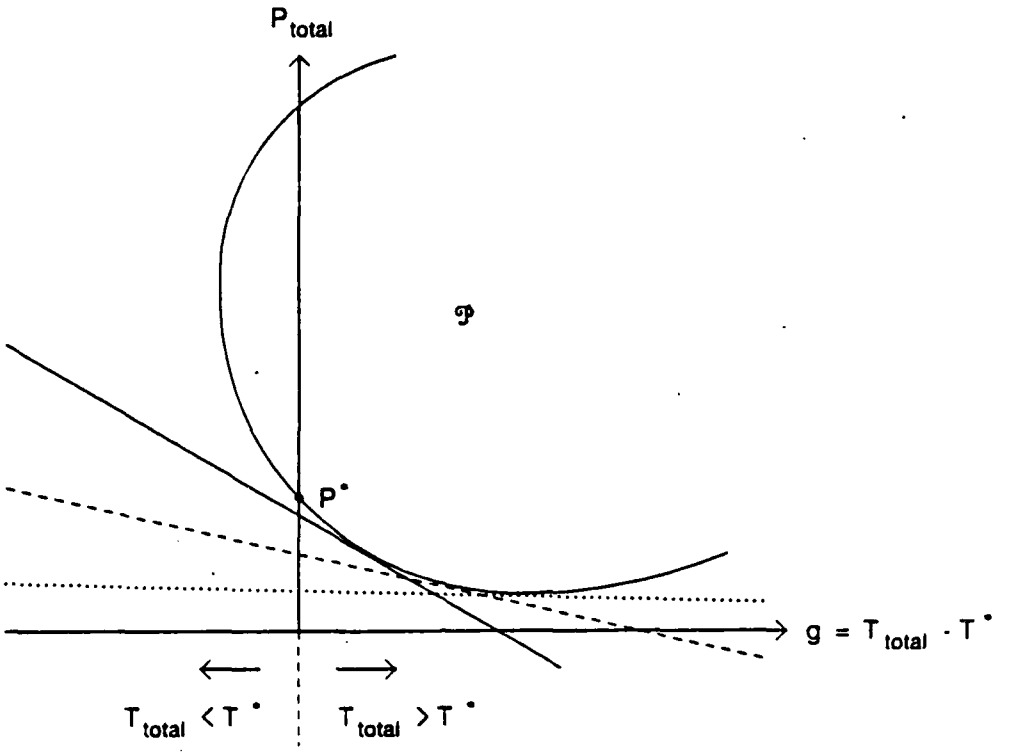


Figure 24: - Outer Loop Maximization

$$P^* = \max \varphi(\mu)$$
$$\text{subject to } \mu \geq 0$$

This maximization gives us the Lagrange multiplier μ of the optimum, while the inner loop minimization provides the optimal transistor size assignments.

We can now grasp the intuitive significance of the Lagrange multiplier. From Figure 24 it is apparent that as μ increases, the line becomes more vertical, and we move up and toward the left. Power consumption increases whereas delay decreases. We are generating transistor size assignments that push the circuit topology harder for speed. The fact that the multiplier has a concrete, practical meaning is quite important, because it allows a designer to follow our CAD tool's "intent" as it optimizes a circuit, showing the signal paths that are the most troublesome in meeting the delay specifications. This knowledge is vital for directing efforts to improve the circuit, such as reduction of interconnect capacitance and modification of circuit topologies (eg. the insertion of super buffers).

4.3.3. Degenerate Cases

It is crucial that optimization algorithms perform properly even when faced with certain degenerate conditions in the delay constraints, such as inactive or infeasible constraints. Inactive constraints can come from one of two sources: (1) a delay specification on a signal path that is so loose that minimum size transistors along the path will satisfy it, or (2) interactions among paths

give rise to a situation where meeting one path's constraint causes another's to be inactive. Of these two possibilities, the second is the most likely, and occurs frequently in practice. An inactive constraint arises because the point of minimum power lies to the left of the constraint's vertical axis in constraint-power space; the dual algorithm will converge to the optimum by driving the constraint's Lagrange multiplier to zero.

Another important degenerate condition comes from infeasible delay constraints, where the designer requests a maximum signal path delay that cannot possibly be met. These cases occur frequently in high-performance circuit design, as the designer pushes a circuit topology and fabrication process to the limits of their performance. Under these situations we desire that the optimizer do the best that it can, sizing those paths with infeasible constraints such that they switch as fast as possible, and sizing paths with feasible constraints such that their power consumption is optimized. The dual algorithm will drive the former paths' Lagrange multipliers towards infinity, sizing the transistors for maximum speed. Thus the algorithm gives useful feedback to the designer, indicating the maximum speed the circuit topology can provide.

4.3.4. Restrictions

As we mentioned at the beginning of our discussion, although duality does offer significant advantages over other optimization methods, it is limited in the scope of objective and constraint functions that it can solve. In particular, certain objective and constraint functions can produce a condition known as a duality gap. These functions give rise to nonconvexities in the lower left boundary of the set $\mathcal{P}$, leading to a gap between the solution found by the dual algorithm and the true optimum P^*. These gaps pose the only potential limitation on the types of circuits that our optimizer can address. We have never encountered a duality gap for any of the circuits we have optimized, nor have we been able to envision any circuit that would produce a significant gap. The power and delay equations describing digital MOS gates, and the separability inherent in the digital MOS domain, make the occurrence of a gap unlikely and imply a small gap even if one should appear. If a gap ever occurs the circuit will still meet delay specifications, with a bounded amount of excess power dissipation.

4.4. Implementation

We have seen that duality maps the nonlinear delay constraints into simple box constraints on Lagrange multipliers. This mapping leads to simple, computationally efficient control structures. The outer loop maximization uses a Davidon-Fletcher-Powell quasi-Newton method [17] with modifications for the box constraints [18]. The inner loop minimization is more complicated since it must handle linear as well as box constraints; it uses an algorithm due to Bard [21]. Since both loops work in small vector spaces and use second derivative information, the optimizer is very fast.

The language chosen to implement the optimizer embodies many of the principles of data abstraction and object oriented programming. These features were essential for supporting our incremental optimization approach. We needed a language that supported automatic dynamic data structure allocation, abstract data types, implicit pointers, and recursive procedure calls and data structures. We chose the CLU programming language [19], which runs on DEC 20s and VAXs. The language system has extensive compile time type checking and an outstanding interactive debugger. Both greatly facilitate program development. Another good choice would have been Zetalisp on a Symbolics 3600. The optimizer consists of generic nonlinear minimization and maximization routines, a circuit optimization support package, and routines for

optimizing generic nMOS logic gates. These program modules represent 3500, 7400, and 2700 lines of CLU code, respectively.

We have applied our optimizer to many circuits; here we present two representative cases. Our first example is a chain of three inverters. (The circuit is small in order to allow a comparison with a general purpose optimizer.) We began with minimum size transistors and requested maximum rise and fall delays of 8.0 ns. The optimizer stops when the delays for active constraints are within five percent of these values. Optimization statistics appear in Table 3. In the table, T_{BEout} is the time until the output begins to move in response to an input transition and T_{SWout} is a measure of how quickly the output switches once it does begin to change. The optimizer reached a solution in slightly over 15 cpu seconds on a DEC System 20/60.

Optimization Accuracy:

	cpu time [sec]		
optimizer	*set up*	*optimization*	*power*
General Purpose (VAX 750)	133.7	3018.0	2.02
Present Work (DEC 20/60)	1.1	15.2	2.08

Delay Accuracies:

	predicted [ns]	*SPICE [ns]*		
Path	T_{BEout}, T_{SWout}	T_{BEout}, T_{SWout}	*error [%]*	*μ [mW/ns]*
in → out, rise	4.06, 3.85	3.80, 3.76	+5	0.403
in → out, fall	5.23, 0.64	5.53, 0.77	−7	0.000

Total SPICE verification time (DEC 20/60): 16.5 cpu sec

Table 3: Optimization Statistics for the Inverter Chain

To verify the accuracy of our optimizer, and also to ascertain the computational speedup afforded by the macromodels and optimization algorithm, we compared its performance to that of a general purpose optimizer. The general purpose optimizer that we used was DELIGHT [1], an experimental optimization package from Berkeley. This package is quite comprehensive, and has been used in applications ranging from designing bridges to fine tuning IC fabrication processes. We interfaced DELIGHT to SPICE, and then attempted to run DELIGHT on the inverter chain. This effort was hampered somewhat by difficulties with DELIGHT's direction finding routine. Owing to the high computational cost of SPICE evaluations, the implementors of DELIGHT were compelled to approximate the derivatives of objective and constraint functions with forward (rather than central) finite differences. While this method requires only half the computation as central finite differences, it is considerably less accurate, and its inaccuracies can cause a direction finder to jam. We had, in fact, originally implemented our gradient routines with forward finite differences and encountered similar problems, but we were able to switch to central differences because of the computational efficiency of our macromodels. To assist DELIGHT's direction finder, we eliminated the maximum beta ratio and minimum shape factor constraints, and started DELIGHT at an initial set of transistor sizes that was fairly close to the optimal solution. The problem was also simplified by not evaluating the chain's rising input, falling output response. This did not affect the final solution since this transition's delay constraint was not active, but it halved the number of SPICE runs needed and simplified the direction finder's task. DELIGHT required five iterations to converge to within five percent of the optimum, consuming

3018 cpu seconds on a VAX 11/750. This time was dominated by SPICE execution. Table 3 gives the statistics.

The performances of the circuits produced by the two optimizers are quite similar. Both have falling input, rising output delays of 8.0 ns as requested, with power consumptions of 2 mW. The power consumption of DELIGHT's circuit is less than ours by about three percent, but this is mainly due to the removal of the minimum S_{pu} constraint on the second inverter.

Our optimizer runs considerably faster than the general purpose optimizer with SPICE. It is difficult to make an exact comparison of how fast DELIGHT would run on the DEC 20/60, had it been able to handle the inverter chain without simplifications, but we can make fairly accurate estimates. A DEC 20/60 will run Fortran code about three or four times faster than will a VAX 750. DELIGHT only evaluated one path transition, with fewer transistor size constraints and an initial set of sizes that was fairly close to the optimum. These simplifications halve the number of SPICE simulations per iteration and reduce the number of iterations needed to reach the optimum, leading to about a factor of five improvement in run time. Hence we believe that DELIGHT would require about 4000 cpu seconds to size the inverter chain on our DEC 20. This is about 300 times slower than our optimizer. We also feel that our run times scale better than those of DELIGHT and SPICE as circuit complexity increases. The partitioning scheme used by our optimizer leads to approximately linear growth, while the growth rates of DELIGHT's feasible directions algorithms and SPICE's simulation algorithms are more rapid.

This comparison of the two optimizers demonstrates a tradeoff of generality and accuracy versus computational speed. DELIGHT is a general purpose optimizer; it provides an extensive framework for solving various kinds of optimization problems. When interfaced to a transistor level simulator such as SPICE, it can size a circuit's transistors to the utmost accuracy possible. In contrast, our optimizer relies on a specialized optimization algorithm and a customized circuit data base, coupled with logic gate macromodels, to achieve a tremendous increase in computational speed. These techniques are simply not applicable to a general purpose optimizer. We feel that our approach is the most appropriate for the transistor sizing problem, but also believe that a general purpose optimizer can play an important role in CAD systems.

We now discuss the optimization of a more complicated example, a four bit adder. One bit of the adder is shown in Figure 25. The adder is comprised of sixteen logic cells having a total of 72 transistors. Path delay constraints were placed on five of the signal paths. Table 4 gives the optimization statistics. Starting with minimum size transistors, the optimizer required only 520 cpu seconds to optimize the adder. In contrast, considerably more time was needed for SPICE runs to just verify the accuracy of the predicted delays.

5. Future Work

At present we are investigating improvements to support a broader class of MOS circuits, such as dynamic logic gates and bootstrapped drivers. These extensions will require writing small software routines to interface the new circuit types to our already existing macromodeling and optimization support packages.

Another improvement would be to integrate the optimizer into a VLSI CAD system. Currently, the optimizer is entirely stand-alone; by integrating it into a CAD system, the optimizer would be more convenient to use. The CAD system we are considering is the Symbolics NS

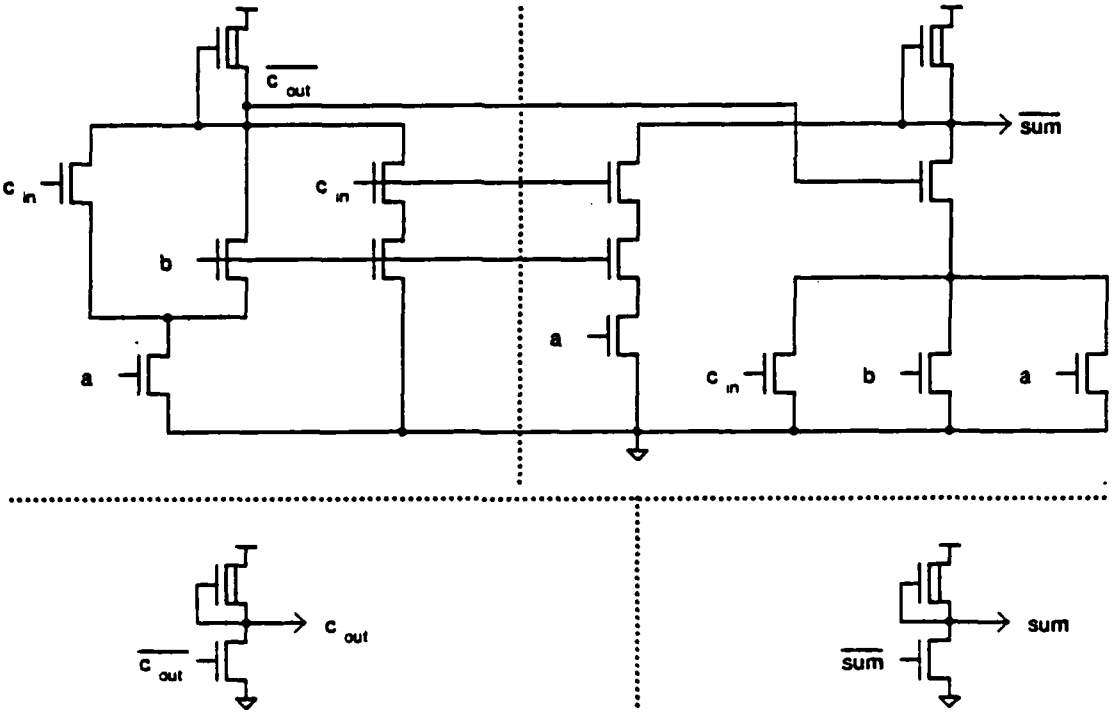


Figure 25: A Full Adder Module

design system [23]. This system supplies extensive support for CAD tools, offering common access among programs to the data objects in the environment and an efficient message passing framework for communication among major CAD subsystems. In addition, the Lisp machine environment in which it is embedded provides a large virtual memory, automatic data allocation and reclamation, and facilities for object oriented programming. Moreover, primarily because the Lisp machine performs data type checking in parallel with computations, the efficiency of the floating point operations would significantly increase relative to the code produced by the CLU compiler for the DEC 20.

We are also studying methods of applying the optimization data to other aspects of circuit design. For example, designers frequently face situations where a speed specification on a circuit path cannot be met, no matter how the circuit's transistors are sized. In such cases designers often try to rearrange interconnect wires and module placements to reduce the capacitive loading on critical nodes, hoping to thereby meet the speed requirement. Our optimizer can be an invaluable asset in such situations. Since it maintains information on the sensitivities of cell delays to input waveforms and capacitive loading, estimates of the new circuit delays after a wiring or module placement change could be provided. Moreover the sensitivities could also be used in conjunction with the values of the Lagrange multipliers to predict the performance of the new circuit after its optimization. These estimates would be extremely useful in guiding a designer during experiments with possible floor plans for a chip.

These concepts can be further extended. Having a tool which can efficiently optimize a

Path	predicted [ns] T_{BEout}, T_{SWout}	SPICE [ns] T_{BEout}, T_{SWout}	error [%]
$A_0 \rightarrow Sum_0$, rise	6.20, 1.66	6.76,1.73	-7
$A_0 \rightarrow Sum_0$, fall	5.53, 0.49	6.20, 0.62	-12
$A_1 \rightarrow Sum_1$, rise	6.26, 1.66	6.67, 1.73	-6
$A_1 \rightarrow Sum_1$, fall	5.53, 0.49	5.96, 0.59	-8
$A_2 \rightarrow Sum_2$, rise	6.28, 1.66	6.72, 1.73	-6
$A_2 \rightarrow Sum_2$, fall	5.52, 0.49	5.92, 0.61	-8
$A_3 \rightarrow Sum_3$, rise	6.19, 1.66	6.78, 1.74	-8
$A_3 \rightarrow Sum_3$, fall	5.44, 0.49	5.86, 0.60	-8
$c_{in} \rightarrow c_{out}$, rise	17.43, 2.46	15.14, 2.55	+12
$c_{in} \rightarrow c_{out}$, fall	18.30, 0.55	18.99, 0.61	-4

Optimization time (DEC 20/60 running CLU):
set up: 2.7 cpu sec optimization: 519.6 cpu sec
Total SPICE verification time (DEC 20/60 running FORTRAN):
1468.6 cpu sec

Table 4: Optimization Statistics for the Four Bit Adder

functional block topology with specified path delays, we can generate rough characterizations of the power-delay tradeoffs for optimized functional blocks. This gives us an aid in topology selection. For instance, the tradeoff curves indicate over which range of delay specifications an adder with a look-ahead carry circuit would be preferable to, say, an adder with a Manchester carry. By integrating these ideas into a VLSI design system, one could form a powerful designer's assistant CAD tool.

An orthogonal but equally important issue concerns the optimizer's implementation. We have gone to considerable pains to preserve the separability inherent in the problem, allowing us to partition it and reap considerable benefits in computational speed. While we originally sought to segment the problem in order to decrease the size of the vector spaces, partitioning also allows us to run the algorithm on parallel computation machines. Recently several workers have developed special purpose parallel machines for VLSI design [24, 25]. These machines have accelerated logic level simulation tremendously. We feel similar success could be had if this optimizer were to be implemented on parallel machines.

6. Acknowledgments

We are pleased to acknowledge the assistance of John Wyatt, Paul Penfield, Jr., and Ron Rivest. Chris Terman, Dimitri Bertsekas, and Rich Zippel read drafts of portions of this paper and gave valuable comments. We are indebted to James Roberge and Rick Carley for valuable discussions on circuit theory, and to Dimitri Bertsekas and Kevin Tsai for introducing us to the art and science of nonlinear optimization. Jack Hilibrand and Larry Rosenberg provided much encouragement and guidance. Jeff Fox, Rich Olsen, and Gerry Sussman were helpful in defining the research topic, while Dan Dobberpuhl and Mark Horowitz kindly shared their knowledge of MOS circuit design and CAD tools. We also thank John Wroclawski, Steve McCormick, and Bob Armstrong for help in implementing this project.

References

[1] W. Nye, E. Polak, A. Sangiovanni-Vincentelli, and A. Tits, "DELIGHT: An Optimization-Based Computer-Aided Design System," *Proceedings International Symposium on Circuits and Systems*, IEEE, April 1981, pp. 851-855.

[2] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated-Circuit Design," *Proceedings of the IEEE*, Vol. 69, No. 10, October 1981, pp. 1334-1362.

[3] A. Ruehli, P. Wolff, and G. Goertzel, "Analytic Power/Timing Optimization Technique for Digital System," *Proceedings 14th Design Automation Conference*, IEEE, June 1977, pp. 142-146.

[4] N. Jouppi, "Timing Analysis for nMOS VLSI," *Proceedings 20th Design Automation Conference*, IEEE, June 1983, pp. 411-418.

[5] S. Trimberger, "Automated Performance Optimization of Custom Integrated Circuits," *Proceedings International Symposium on Circuits and Systems*, IEEE, May 1983, pp. 194-197.

[6] J. Ousterhout, "Switch-Level Delay Models for Digital MOS VLSI," *Proceedings 21st Design Automation Conference*, IEEE, June 1984, pp. 542-548.

[7] M. Horowitz, *Timing Models for MOS Circuits*, PhD dissertation, Stanford, 1983.

[8] L. P. J. Hoyte, "Automated Calculation of Device Sizes for Digital IC Designs," Master's thesis, MIT, 1982.

[9] L. Glasser and L. Hoyte, "Delay and Power Optimization in VLSI Circuits," *Proceedings 21st Design Automation Conference*, IEEE, June 1984, pp. 529-535.

[10] J. Roberge, *Operational Amplifiers: Theory and Practice*, Wiley, 1975, pp. 97-104.

[11] M. Matson, *Macromodeling and Optimization of Digital MOS VLSI Circuits*, PhD dissertation, MIT, January 1985.

[12] P. Gray and C. Searle, *Electronic Principles: Physics, Models, and Circuits*, Wiley, 1969.

[13] J. Meyer, "MOS Models and Circuit Simulation," *RCA Review*, Vol. 32, March 1971, pp. 42-63.

[14] W. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, Vol. 19, No. 1, January 1948, pp. 55-63.

[15] J. Rubinstein, P. Penfield, and M. Horowitz, "Signal Delays in RC Tree Networks," *IEEE Transactions on Computer Aided Design*, Vol. CAD-2, No. 3, July 1983, pp. 202-211.

[16] T.-M. Lin and C. Mead, "Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits," *Proceedings, Conference on Advanced Research in VLSI*, MIT, January 1984, pp. 93-99.

[17] R. Fletcher and M. Powell, "A Rapidly Convergent Descent Method for Minimization," *Computer Journal*, Vol. 6, 1963, pp. 317-322.

[18] D. Bertsekas, "Projected Newton Methods for Optimization Problems with Simple Constraints," *SIAM Journal on Control and Optimization,* Vol. 20, 1982, pp. 221-246.

[19] B. Liskov, A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU," *Communications of the ACM,* Vol. 20, No. 8, August 1977, pp. 564-576.

[20] D. Luenberger, *Introduction to Linear and Nonlinear Programming,* Addison-Wesley, 1984, second edition.

[21] Yonathan Bard, *Nonlinear Parameter Estimation,* Academic Press, 1974.

[22] D. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods,* Academic Press, 1982.

[23] J. Cherry, H. Shrobe, N. Mayle, C. Baker, H. Minsky, K. Reti, and N. Weste, "NS: An Integrated Symbolic Design System," *Proceedings VLSI 85,* International Federation for Information Processing, August 1985, pp. 319-328.

[24] G. Pfister, "The Yorktown Simulation Engine: Introduction," *Proceedings 19th Design Automation Conference,* IEEE, June 1982, pp. 51-54.

[25] Zycad Corporation, *LE-1000 Series Logic Evaluator Intermediate Form Specification,* Roseville, Minnesota, 1983, Release 1.0.

END

DTIC

5-86